P.1877/91

6 1991

informatyka

Komputery w medycynie

WYDAWNICTWO SIGMA NOT

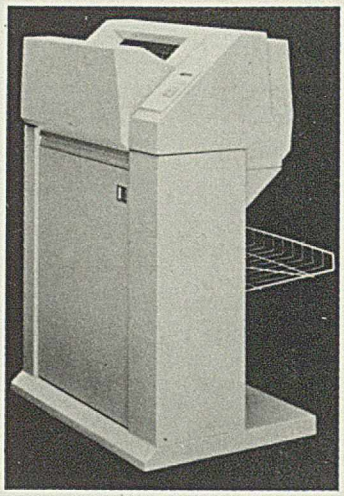
Cena 12 000 zł

SUPRA SP. Z O.O.
ul. Trybunalska 54, 60-325 POZNAŃ
telefon: 674392, telefaks: 236498
teleks: 412966

AUTORYZOWANY DYSTRYBUTOR FIRMY MANNESMANN TALLY OFERUJE DRUKARKI HEAVY DUTY

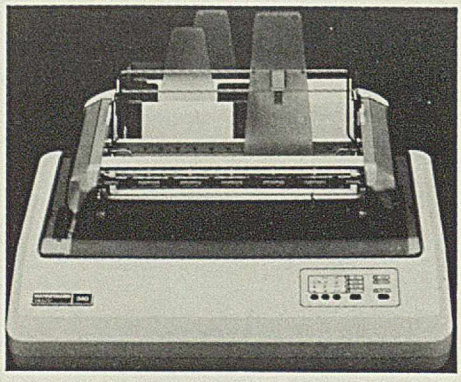
MT 645
drukarka mozaikowo-wierszowa

- doskonała jakość druku
- niezawodność 7000 h MTBF
- duża szybkość 450 wierszy/min.
- cicha praca
- bufor 27 KB
- liczba kopii 1 + 5
- pełna grafika 240 (dpi)
- polskie litery (opcja)
- wewnętrzny program testujący
- dwa interfejsy automatycznie przełączane
- podwójny system traktorów



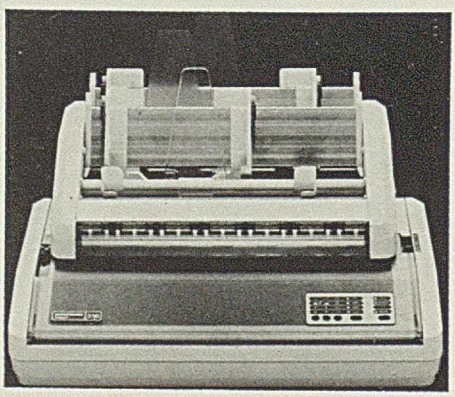
MT 340
drukarka mozaikowa (18 igłowa)

- doskonała jakość druku
- niezawodność
- szybkość 400 zn/sek.
- cicha praca
- polskie litery (opcja)
- druk kolorowy (opcja)



MT 230
drukarka mozaikowa (9, 18, 24 igłowa)

- doskonała jakość druku
- niezawodność
- szybkość 300 zn/sek.
- cicha praca
- polskie litery (opcja)
- druk kolorowy (opcja)



MANNESMANN TALLY

PRAWDOPODOBNIE NAJLEPSZE DRUKARKI NA ŚWIECIE!

0/2/91

informatyka

Miesięcznik
ISSN 0542-9951
INDEKS 36124

P.1877/91

nr 6 1991
Czerwiec
Rok wydania XXVI

KOLEGIUM REDAKCYJNE:

mgr Jarosław DEMINET
dr inż. Wacław ISZKOWSKI
mgr Teresa JABŁOŃSKA
(sekretarz redakcji)
Władysław KLEPACZ
(redaktor naczelny)
dr inż. Wojciech MOKRZYCKI
mgr inż. Jan RYŻKO
mgr Hanna WŁODARSKA

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab.
Juliusz Lech KULIKOWSKI

WYDAWCA:
Wydawnictwo Czasopism i Książek Technicznych SIGMA NOT
Spółka z o.o.
ul. Biała 4
00-950 WARSZAWA
skrytka pocztowa 1004

Redakcja:
01-552 Warszawa,
Pl. Inwalidów 10, p. 104, 105
tel. 39-14-34

Materiałów nie zamówionych redakcja nie zwraca

W sprawach ogłoszeń prosimy zwracać się bezpośrednio do Redakcji lub Działu Reklamy i Marketingu 00-950 Warszawa ul. Biała 4 telefon: 20-31-24 telefaks: 20-31-16 teleks: 814550

W numerze:

Strona

Dlaczego UNIX nie jest systemem czasu rzeczywistego? – *Krzysztof Nałęcki* 2
Metody implementacji systemów czasu rzeczywistego – *Wojciech Trojniar* 8
Środowisko do projektowania prostych systemów czasu rzeczywistego – ćwiczenia w programowaniu obiektowym – *Paweł Gizbert-Studnicki, Jerzy Karczmarczuk* 12
Rola komputera w ocenie estetycznej kształtów – *Ewa Grabska, Agnieszka Polarska* 17

/unix

Okna w UNIX-ie: System X i jego otoczenie – *Tomasz Wójtowicz* 22
O standardzie języka C (3) – *Władysław Majerski* 25
/mail 26

W najbliższych numerach:

Lipcowy numer poświęcono w całości zastosowaniom komputerów w medycynie.

- Juliusz Lech Kulikowski prezentuje komputerową analizę obiektów i rekonstrukcję relacji.
- Jerzy Cytowski omawia system przetwarzania i analizy sygnałów EKG o dużym wzmocnieniu.
- Piotr J. Jasiński, Wojciech Ticner i Marcin Januchta dokonują przeglądu rozwoju standardu dokumentacji medycznej dla systemów eksperckich oraz podają przykłady tego standardu w różnych specjalizacjach medycznych.
- Konrad Łukaszewicz i Zbigniew Mejran zajmują się syntezatorami narządu mowy, słuchu lub wzroku.
- Marek Kurzyński, Jerzy Sas i Edward Puchała opisują trzy niekonwencjonalne metody rozpoznawania schorzeń za pomocą komputerowej diagnostyki medycznej.
- Anna Zawada charakteryzuje problemy związane z automatycznym czytaniem pisma, np. przez osoby niewidome.
- Leon Bobrowski opisuje komputerowy system wspomagania diagnostyki oraz analizy danych „Hepar".
- Robert Rudowski i Maciej Guć – omawiają systemy eksperckie sztucznej wentylacji płuc.

Warunki prenumeraty

Przyjęcie prenumeraty – wyłącznie na podstawie dokonanej wpłaty na drukach dostarczanych dotychczasowym prenumeratorom przez Wydawnictwo, lub nowym – po uprzednim zgłoszeniu zapotrzebowania (pisemnie lub telefonicznie) w Zakładzie Kolportażu Wydawnictwa.

Blankiet wpłaty – powinien zawierać następujące informacje: dokładna nazwa i adres (z kodem pocztowym) zamawiającego, tytuły zamawianych czasopism, ich liczbę i okres prenumeraty.

Wpłata – zgodnie z podanymi cenami należy dokonać w banku lub w UPT na konto podane na naszym blankiecie, tj:
Państwowy Bank Kredytowy III O/Warszawa nr: 370015-1573-139-11

Prenumeratorzy zbiorowi – osoby prawne obowiązują blankiety „Wpłata-Zamówienie". Cena normalna.

Prenumeratorzy indywidualni – osoby fizyczne obowiązują blankiety typu przekazy dla wpłat na rachunki bankowe. Cena normalna.

Prenumerata ulgowa – zgodnie z podaną ceną ulgową przysługuje wyłącznie osobom fizycznym, będącym członkami SNT, studentom i uczniom szkół zawodowych. Uczniowie szkół ogólnokształcących mogą zamówić w prenumeracie ulgowej tylko miesięcznik „Aura"

Uwaga! w podanym okresie prenumeraty można zamówić tylko po jednym egzemplarzu z każdego tytułu.

Prenumerata ze zleceniem wysyłki za granicę – cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa od ceny normalnej.

Należy podać dokładny adres odbiorcy za granicą.

Terminy przyjmowania prenumeraty:
– do 10 listopada na I, II, III, IV kwartał następnego roku
– do 28 lutego na II, III, IV kwartał br.
– do 31 maja na III i IV kwartał br.
– do 31 sierpnia na IV kwartał br.

Zmiany w prenumeracie, np. zmiana liczby tytułów, liczby egzemplarzy, rezygnacja z prenumeraty, można zgłaszać tylko w podanych terminach z mocą obowiązującą od następnego kwartału.

Egzemplarze archiwalne (z lat ubiegłych)
Można nabyć za gotówkę w Klubie Prasy Technicznej, Warszawa, ul. Mazowiecka 12 (tel. 26-80-16) lub zamówić pisemnie w Zakładzie Kolportażu, Dział Handlowy, 00-950 Warszawa, skr. poczt. 1004 (tel. 40-37-31), na rachunek lub za zaliczeniem pocztowym.

Informacji o prenumeracie udziela: Zakład Kolportażu Wydawnictwa SIGMA-NOT Spółka z o.o., 00-716 Warszawa, ul. Bartycka 20, skr. 1004. Telefony: 40-00-21 wewn. 293, 295, 299 lub 40-30-86, 40-35-89.

Wstępna cena jednego egzemplarza na 1991 rok: normalna – 12 000 zł ulgowa – 8400 zł

Wartość prenumeraty (w zł):
Normalna: kwartalna – 36 000, półroczna 72 000, roczna 144 000
Ulgowa: kwartalna – 25 200, półroczna 50 400, roczna 100 800

Uwaga: W przypadku zmiany cen w okresie objętym prenumeratą, prenumeratorzy zobowiązani są do dopłaty różnicy cen.

Skład i druk: Drukarnia SIGMA NOT Sp. z o.o. z. 180/91 n. 3650 egz.

Trzy pierwsze artykuły zostały przygotowane na Drugą Wiosenną Szkołę PTI Oddziału Górnośląskiego (Szczyrk '90). Artykuł Krzysztofa Nałęckiego informuje dlaczego UNIX nie jest systemem czasu rzeczywistego i przedstawia jednocześnie wymogi, jakie system taki powinien spełniać. Tekst autorstwa Wojciecha Trojniara prezentuje konkretne implementacje systemów czasu rzeczywistego, a opracowanie Pawła Gizberta-Studnickiego i Jerzego Karczmarczuka omawia zalety programowania obiektowego zastosowania do projektowania systemów czasu rzeczywistego.

POLSKIE TOWARZYSTWO INFORMATYCZNE

KRZYSZTOF NAŁĘCKI
Instytut Informatyki
Politechnika Śląska
Gliwice

Dlaczego UNIX nie jest systemem czasu rzeczywistego?

Praca w czasie rzeczywistym jest charakterystyczna dla komputera „zanurzonego" (ang. *embedded*) w zastosowaniu. Pewną analogią może tu być użycie silnika elektrycznego w wiertarce, drukarce czy wideoodtwarzaczu – jest konieczny do pracy urządzenia, ale póki działa poprawnie jest dla użytkownika „niewidoczny".

Zadaniem oprogramowania tak zastosowanego komputera (obecnie skonstruowanego najczęściej na bazie mikroprocesora) jest nadzorowanie pracy i sterowanie urządzeniami nie przypominającymi zwykle standardowych urządzeń peryferyjnych. Zdarzenia w otoczeniu komputera, dotyczące nadzorowanych urządzeń, są przekazywane w formie przerwań (pobudzeń) inicjujących akcje realizowane przez oprogramowanie. System jest zatem pobudzany zdarzeniami.

Mimo że wspomniane zdarzenia mogą pojawiać się w dowolnych momentach czasu, wymaga się, by reakcje systemu (oprogramowania) były wykonywane w założonych i nieprzekraczalnych terminach, i to w każdym, nawet „najgorszym" przypadku. Stąd termin *system czasu rzeczywistego*.

W systemie czasu rzeczywistego wynik uzyskany zbyt późno jest tak samo zły i niedopuszczalny jak wynik błędny. Wyrażając to krócej, zadaniem systemu czasu rzeczywistego jest wytworzenie przewidywalnej reakcji na zdarzenie zewnętrzne w założonym odcinku czasu, nawet w najgorszym przypadku. Przewidywalność czasu reakcji jest najważniejszą cechą takiego systemu. Nie wystarczy znać wskaźnik MIPS procesora, by mieć pewność, że grzejnik zostanie wyłączony zanim mleko wykipi.

W zastosowaniach bardziej klasycznych, np. w systemie wielodostępnym, użytkownik na pewno będzie bardzo zadowolony, gdy system zareaguje na naciśnięcie klawisza z opóźnieniem 100 ms i nie zauważy różnicy, gdy opóźnienie to zwiększy się do 200 ms. W rzeczywistości, wystarczy świadomość, że system zareaguje w „najbliższym czasie", a użytkownik nie zapłonie, wybuchnie lub złamie się (choć czasem jego system nerwowy może być poddany – metaforycznie rzecz ujmując – podobnym stresom).

System czasu rzeczywistego musi być ponadto analizowany w kategoriach najgorszego przypadku – największego obciążenia. Wielkość średniego czasu reakcji nie ma żadnego znaczenia w sytuacji, może i bardzo mało prawdopodobnej, wprowadzenia opóźnienia w podniesienie sterów wysokości w samolocie zbliżającym się do pionowej przeszkody. Tak jak i przysłowiowy saper – w niektórych zastosowaniach system czasu rzeczywistego może się spóźnić najwyżej raz.

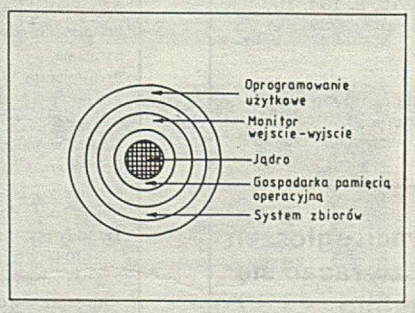
Struktura systemu

Najprostszą formą organizacji systemu operacyjnego komputera jest struktura monolityczna, niepodzielna. W takim systemie fragmenty realizujące funkcje systemu – zarządzanie zasobami i procesami[1)] są trudne do wyróżnienia dla obserwatora zewnętrznego. Gospodarka czasem procesora, zarządzanie pamięcią, operacje wejścia–wyjścia, dostęp do zbiorów danych są wykonywane jako niepodzielne operacje, zwykle w wyróżnionym, uprzywilejowanym trybie pracy procesora.

W przypadku pracy przy wymaganiu reakcji w czasie rzeczywistym, ze względu na asynchroniczność i nieprzewidywalność, zdarzeń w otoczeniu systemu i równoczesną współbieżność wielu realizowanych procesów, musimy odrzucić strukturę monolityczną jako zbyt prostą. Konieczne jest przyjęcie struktury dekomponowalnej (podzielnej), pozwalającej na rozróżnianie akcji systemu operacyjnego i sterowanie ich przebiegiem.

Najczęściej stosowanym modelem jest struktura hierarchiczna, a najniższy (najbardziej wewnętrzny) jej poziom nazywany jest jądrem systemu (ang. *kernel*). Jądro realizuje podstawowe niskopoziomowe funkcje systemu.

Ten minimalny zestaw funkcji jest używany do konstrukcji poziomu następnego i tak warstwowo konstruuje się kompletny system. W końcu, na system operacyjny „nawleka" się oprogramowanie użytkowe (rys. 1).



Rys. 1. Struktura systemu operacyjnego czasu rzeczywistego

Cechy jądra systemu

Jądro systemu operacyjnego czasu rzeczywistego powinno spełniać następujące wymagania:

- **Współbieżność.** Asynchroniczność zdarzeń w otoczeniu systemu dyktuje współbieżne (równoczesne wykonywanie, być może wielu, procesów obsługi tych zdarzeń. Zadaniem jądra jest przełączanie

[1)] Proces – wykonanie programu, obiekt dynamiczny o skończonym czasie życia ubiegający się o przydział zasobów komputera.

procesora między procesami, adekwatnie do istniejącej w systemie sytuacji i postawionych wymagań.

- **Wywłaszczalność.** Przy pracy w trybie czasu rzeczywistego konieczne jest wprowadzenie priorytetów ustalających wzajemną ważność wykonywanych akcji. Proces o wysokim priorytecie, gotowy do wykonania (tzn. istniejący i nie zablokowany w oczekiwaniu na jakieś zdarzenie lub sygnał) musi uzyskać dostęp do procesora, wywłaszczając z niego każdy inny proces o niższym priorytecie.
- **Szybkie przełączanie procesora (kontekstu).** Przekazanie procesora procesowi o najwyższym priorytecie (tzw. przełączenie kontekstu) musi być wykonywane z minimalnym narzutem czasu ze strony jądra. W systemach wielodostępnych (np. UNIX) czas przełączania sięga dziesiątków milisekund, w systemach operacyjnych czasu rzeczywistego wymaga się czasu przełączania rzędu dziesiątków mikrosekund.
- **Małe opóźnienie w przyjęciu zgłoszenia** (przerwania). Zgłoszenie o zdarzeniu w otoczeniu systemu jest przekazywane za pośrednictwem przerwania. Wymaga się, by czas jaki mija od pojawienia się przerwania do rozpoczęcia jego obsługi był rzędu kilkunastu mikrosekund. Nakłada to ostre ograniczenia na długość odcinka czasu, w którym procesor pracuje z zablokowanym układem przerwań. Szybkie przełączenie kontekstu (np. w czasie 15 μs) w sytuacji, gdy system blokuje układ przerwań na czas 100 μs jest mało przydatne.
- **Synchronizacja.** Współbieżnie wykonywane procesy muszą ze sobą kooperować, gdyż ich akcje dotyczą tego samego obiektu. Kooperacja ta może być realizowana na poziomie wzajemnej synchronizacji procesów, a jądro jest zobowiązane ten mechanizm udostępnić i realizować.
- **Komunikacja.** Komunikacja między procesami jest następnym poziomem umożliwienia im kooperacji. Komunikacja jest pojęciem szerszym od wspomnianej wyżej synchronizacji – tę ostatnią można zrealizować przesyłając komunikaty „binarne" – TAK NIE.
- **Wydajność.** W systemie czasu rzeczywistego kluczowym parametrem jest graniczna wydajność systemu, uzyskiwana przy najbardziej niekorzystnej kompozycji zgłoszeń i obciążenia. System, w którym określona funkcja jest realizowana zawsze w czasie nie dłuższym od 100 μs jest o wiele lepszy od systemu, w którym ta sama funkcja jest wykonywana średnio w czasie 20 μs, ale nie można gwarantować jej wykonania w czasie nie dłuższym od 200 μs (jak to odbywa się np. w dość typowym rozkładzie wykładniczym czasu obsługi).
- **Wymagania specjalne.** Wymagania te wynikają bądź ze specyfiki zastosowania – np. system wieloprocesorowy, bądź coraz częściej z rozwoju możliwości technicznych – np. procesory typu RISC.

Jądro a system operacyjny

Termin „jądro" jest używany w dziedzinie systemów operacyjnych już od długiego czasu i stał się pojęciem klasycznym. Najczęściej jednak w zakres funkcji jądra jego konstruktorzy włączają również gospodarkę pozostałymi zasobami systemu: przydział procesom pamięci operacyjnej (być może na niskim poziomie), obsługę operacji wejścia–wyjścia czy odmierzanie czasu (zegar też zgłasza się za pośrednictwem przerwania). Tego typu rozwiązanie należy traktować raczej jako „prosty" system o strukturze monolitycznej, uzupełnianej przez procesy tzw. systemowe.

Pewnym kryterium może być właśnie fakt wyróżniania procesów systemowych i ich preferowanie w stosunku do procesów użytkowych. W systemie czasu rzeczywistego taki podział nie ma uzasadnienia. Często, procesy wykonywane według programów tzw. użytkowych, napisanych nie przez autorów systemu operacyjnego, muszą mieć wyższy priorytet od standardowych funkcji systemu operacyjnego.

Poziom jądra

W każdym systemie wielozadaniowym właściwe przetwarzanie danych jest wykonywane przez procesy – realizacje dostarczanych programów. Czasem jednak, np. w okresie przełączania kontekstu, nie można wskazać procesu, do którego procesor w tym momencie należy. W takich sytuacjach procesor jest wykorzystywany właśnie przez jądro, a wykonywane w tym czasie akcje są realizowane na poziomie jądra.

System operacyjny (jądro) nie zezwala na wywłaszczenie siebie z procesora w czasie pracy na poziomie jądra, bo kto w tej sytuacji mógłby przejąć procesor?

Wyobraźmy sobie, że jądro przekazuje procesor z procesu *MAŁY* do procesu *ŚREDNI* (zgodnie z ich priorytetami) i w tym czasie wystąpiło zdarzenie uaktywniające proces *DUŻY*, o najwyższym priorytecie. Najpierw musi nastąpić całkowite przełączenie kontekstu dla procesu *ŚREDNI*, dopiero po tym może zostać przyjęte przerwanie aktywizujące proces *DUŻY* i ponowne przełączenie kontekstu z procesu *ŚREDNI* na proces *DUŻY*.

W czasie pracy na poziomie jądra przyjmowanie przerwań musi być zablokowane. Z tego powodu oczywista jest tendencja do ograniczania liczby i zakresu funkcji realizowanych przez jądro. Jako minimalny zestaw funkcji można zaproponować:

- podstawowy poziom gospodarki procesami, tzn. przełączenie kontekstu zgodnie z priorytetem procesów;
- podstawowe mechanizmy synchronizacji, np. semafory binarne.

Zaproponowany zestaw umożliwi realizację pozostałych funkcji systemu, o czym postaram się przekonać Czytelnika w dalszym toku rozważań.

Warto chyba w tym miejscu zauważyć, że system operacyjny UNIX nie spełnia wymienionych wyżej postulatów. Zbyt duża część funkcji systemu jest realizowana w formie akcji nieprzerywalnych, a więc na poziomie jądra (w tym przypadku niezależnie od jego zdefiniowania, a ponadto, UNIX zdecydowanie preferuje „swoje", tzn. systemowe procesy.

Realizacja

Nie pretendując do próby przedstawienia kompletnego projektu kolejnego, „uniwersalnego" systemu czasu rzeczywistego spróbuję wykazać, w jaki sposób można zorganizować taki system, by spełniał on przedstawione wyżej postulaty.

Realizacja funkcji systemowych

Wymaganie małego opóźnienia w odbiorze przerwań wymusza konieczność realizacji większości funkcji systemowych jako procesów przerywalnych i wywłaszczalnych – z konkretną wartością priorytetu, zwykle równą lub niewiele wyższą od priorytetu procesu wywołującego funkcję. Funkcja systemowa jest realizowana podobnie do wywołania „zwykłej" procedury bibliotecznej.

Używane powszechnie do aktywizacji funkcji systemowych wywołanie przerwania programowego, a więc przejście na poziom jądra (jak po każdym przerwaniu), służy w tym przypadku wyłącznie do utworzenia nowego procesu z określoną wartością priorytetu i zablokowania procesu wywołującego do chwili zakończenia tego nowego procesu. Ponieważ i te operacje mogą zająć zbyt dużo czasu, niech je wykona „nadproces", tzn. proces o jeszcze wyższym priorytecie.

Od jądra oczekiwać wystarczy realizacji następujących operacji:

```
twórz_proces (...)
usuń_proces (ID)
zawieś_proces (ID)
odwieś_proces (ID)
priorytet_procesu (ID, prior)
```

W ramach akcji realizowanych w obrębie funkcji systemowych należy spodziewać się wystąpienie tzw. sekcji krytycznych – należy spodziewać się wystąpienia tzw. sekcji krytycznych – fragmentów kodu, które muszą być wykonane jako operacja niepodzielna (np. przydział zasobu). By nie wykonywać sekcji krytycznej na poziomie jądra wystarczy dysponować semaforami binarnymi i objąć odpowiedni fragment kodu operacjami semaforowymi w sposób następujący:

```
        ...
    kod_niekrytyczny
        ...
P (semafor_tej_sekcji);
        ...
    sekcja_krytyczna_tej_operacji;
        ...
V (semafor_tej_sekcji);
        ...
    kod_niekrytyczny
        ...
```

Jądro musi zatem udostępnić procesom standardowe operacje semaforowe P ( ) i V ( ) (nazwy wg twórcy pojęcia – E. Dijkstry).

Zarządzanie procesami

Głównym zadaniem jądra jest zarządzanie procesami, tzn. przekazanie jednostki centralnej jednemu z gotowych do wykonania procesów, po zakończeniu operacji na poziomie jądra (obsłudze przerwania). Sprowadza się to do wyboru procesu-następcy oraz takiego przełączania procesów, by proces uruchomiony miał kompletne środowisko (zasoby) w stanie, w którym poprzednio odebrano mu jednostkę centralną. Stan środowisk procesów unieruchomionych musi być w taki sposób zapamiętany, by w przyszłości można było go odtworzyć.

W określonej chwili proces może być:
- **wykonywany**; proces korzysta z jednostki centralnej; w systemie z jednym procesorem tylko jeden proces może być w tym stanie;
- **gotowy**; proces może być uruchomiony, jeśli jądro wybierze ten proces na swego następcę;
- **czekający**; proces nie może być uruchomiony, bo oczekuje na wystąpienie określonego zdarzenia (np. wykonania przez inny proces operacji V ( ) na semaforze), po wystąpieniu którego jego stan zmieni się na „gotowy";
- **zawieszony**; proces został wyłączony, a po włączeniu („odwieszeniu") przez inny proces przejdzie do stanu, w jakim znajdował się w chwili zawieszania (gotowy lub czekający).

Rozwiązanie problemu gospodarki procesami sprowadza się do określenia kryterium wyboru jednego z gotowych procesów oraz struktur danych opisujących procesy, umożliwiających wykonanie efektywnego wyboru.

W systemie czasu rzeczywistego jedyne akceptowalne kryterium wyboru to regulamin priorytetowy z priorytetami bezwzględnymi. Jego istota sprowadza się do tego, że w każdej chwili proces wykonywany to ten proces, który pośród procesów gotowych ma priorytet najwyższy.

Jeżeli założymy niezmienność priorytetów, strukturą organizującą dane może być stos. Jeśli natomiast dopuścimy zarówno zmienność priorytetów, jak i zawieszanie procesów gotowych, to stos nie może być brany pod uwagę.

Kolejnym kandydatem jest zatem lista. Lista ta zawiera wykaz (identyfikatory) procesów i jest uporządkowana według wartości ich priorytetów. Ponieważ jednak czas przeglądania listy jest liniowo zależny od jej długości, można zaproponować następujące usprawnienia:
- po akcji na poziomie jądra, która mogła zmienić skład podzbioru procesów gotowych lub priorytet któregoś z procesów („czekający" → „gotowy", „zawieszony" → „gotowy", „zmień priorytet na wyższy"), listę procesów należy przeglądać od początku, poszukując procesu gotowego o bezwzględnie najwyższym priorytecie;
- po akcji na poziomie jądra, która nie spowodowała wyżej wymienionych skutków, procesor jest zwracany procesowi, który znajdował się w stanie „wykonywany" przed rozpoczęciem pracy jądra;
- po akcji na poziomie jądra, spowodowanej przez proces wykonywany i prowadzącej do jego zawieszenia, wprowadzenia w stan czekania lub zakończenia procesu, lista jest przeglądana od miejsca, które zajmuje (zajmował) proces ostatnio wykonywany;
- dla skrócenia czasu przeglądania listy można ją zorganizować jako listę (pod)list; „głowa" podlisty procesów o równej (lub zbliżonej) wartości priorytetu, występująca w liście głównej, informuje o długości podlisty i liczbie procesów gotowych na tym poziomie.

W przypadku podlisty obejmującej grupę procesów o tym samym priorytecie realizuje ona funkcję naturalnej kolejki. Gdy podlista obejmuje grupę procesów o różnych (zbliżonych) priorytetach można zastosować ten sam pomysł, aż do uzyskania w efekcie zapisu danych o procesach w strukturze drzewa (tzn. drzewo 2/3). Dla takiego rozwiązania lokalizacja elementu zajmuje nie więcej niż $\log_2 n$ kroków. Jednak w strukturze takiej jest kosztowna aktualizacja (w sensie czasu).

Opis środowiska procesu

Opis środowiska procesu objąć musi te dane, które są konieczne do prawidłowego przebiegu procesu po przydzieleniu mu jednostki centralnej, a więc stan tej jednostki. W najprostszym przypadku wystarczy zapamiętanie rejestrów określających stan. Sprawa komplikuje się tym bardziej, im więcej jest zaawansowana architektura procesora, np. im więcej w niej występuje rejestrów, w tym rejestrów arytmetycznych (procesory typu RISC). Rośnie bowiem czas zapamiętania i odtworzenia zawartości tych rejestrów oraz rozmiar zużywanej w tym celu pamięci operacyjnej.

Podobnie wygląda problem w przypadku wirtualizacji pamięci operacyjnej oraz buforowania tej pamięci w pamięci podręcznej; przełączenie kontekstu związane jest z „przestawieniem" mechanizmów dostępu do pamięci operacyjnej na mechanizmy właściwe dla uruchamianego procesu.

W opisie środowiska procesu muszą ponadto znaleźć się dane reprezentujące pozostałe przydzielone mu zasoby (pamięć operacyjna, urządzenia we–wy). Dla procesów w stanie oczekiwania konieczne jest powiązanie z przyczyną oczekiwania (np. semafora).

Proces, rozpoznawany przez jądro na podstawie indywidualnego identyfikatora (numeru) jest reprezentowany zatem przez swój stan, priorytet i środowisko.

Niebezpieczeństwo blokady wzajemnej

Istnienie zasobów niewywłaszczalnych prowadzi do potencjalnej możliwości powstania blokady wzajemnej procesów przy „nieostrożnej" gospodarce zasobami. W systemie czasu rzeczywistego próg wywłaszczalności, tzn. czas niezbędny do skoordynowanego odebrania zasobu bez utraty danych (zapamiętanie stanu zasobu), jest zwykle bardziej ostry.

Zasobem niewywłaszczalnym może stać się nawet obszar pamięci operacyjnej, wtedy gdy wymagania na czas reakcji nie zezwalają na wymianę zawartości między pamięcią operacyjną i pamięcią dyskową, przy czym wymaganie takie może dotyczyć wybranej grupy procesów, zwykle o dużej wartości priorytetu.

Źródłem blokady mogą stać się również semafory, a ściślej, sekcje krytyczne na których „straży" umieszczono semafory.

Ochronę przed blokadą wzajemną musi zapewnić system operacyjny, a więc wmontowane weń algorytmy gospodarki zasobami. Stosować można tu wszystkie znane techniki, w tym hierarchizację zasobów lub nawet zapewnienie takiej liczby zasobów krytycznych, aby blokada nie mogła zaistnieć.

W przypadku semaforów, najprostsza, elementarna technika ochrony przed wzajemnym zablokowaniem się procesów na (co najmniej!) parze semaforów, narzuca konieczność wprowadzenia wymagania, aby nie mogły nakładać się dwie sekcje krytyczne. Inaczej mówiąc, nie wolno wykonywać operacji P ( ) przed zwolnieniem innego semafora użytego wcześniej operacją V ( ).

Inne zasoby

Jądro jest odpowiedzialne za gospodarkę procesami, a więc gospodarkę czasem procesora. System operacyjny musi zapewnić poprawną, tzn. bezblokadową gospodarkę wszystkimi zasobami. Pozostałe zasoby to pamięć operacyjna, urządzenia wejścia–wyścia oraz zbiory danych (dyskowe). Te ostatnie mogą nie występować w dedykowanym, „zanurzonym" systemie komputerowym.

Gospodarkę tymi pozostałymi zasobami „wynieśliśmy" już poprzednio poza jądro. Będzie ona wykonywana na poziomie procesów uruchamianych w ramach systemu operacyjnego. Z punktu widzenia jądra procesy takie nie wyróżniają się niczym szczególnym, poza – być może – wysokim poziomem priorytetu.

Natomiast obserwując system z zewnątrz możemy zauważyć pozorną „nieśmiertelność" procesów gospodarki zasobami. Kreowane przy zapoczątkowaniu pracy systemu, przebywają one głównie w stanie oczekiwania lub zawieszenia. Aktywizowane przez akcje jądra, wynikające z działań innych procesów lub przerwania zewnętrzne (np. obsługa urządzeń we–wy), po wykonaniu funkcji powracają do stanu letargu.

W systemie operacyjnym czasu rzeczywistego cechę „nieśmiertelności" mogą mieć również wybrane procesy użytkowe, a ich odróżnianie od procesów systemowych jest sprawą umowy.

Pamięć operacyjna

Pamięć operacyjna jest następnym, po czasie procesora, zasobem koniecznym dla realizacji procesu. Przydzielone obszary (lub obszar) pamięci zawierają wykonywany kod i przetwarzane dane. W przypadku dysponowania pamięcią masową o dostatecznie krótkim czasie dostępu, pamięć operacyjną można uznać za za zasób wywłaszczalny, symulując ją w obszarze zbioru dyskowego. Jednak przy ostrych wymaganiach czasowych, narzut wprowadzany przesyłaniem z (i) do pamięci dyskowej może zmusić do uznania pamięci operacyjnej za zasób niewywłaszczalny dla określonych procesów lub wymusić konieczność użycia pamięci zewnętrznej o szybkości porównywalnej z szybkością pamięci operacyjnej (np. dysk typu RAM).

W trakcie realizacji procesu może pojawiać się ponadto potrzeba zmiany rozmiaru pamięci danych tego procesu, wówczas w wyniku przetwarzania liczba danych rośnie lub maleje albo w krótkim czasie jest potrzebny dodatkowy blok pamięci do przechowywania wyników pośrednich.

Obszar pamięci przydzielony określonemu procesowi musi być ponadto technicznie niedostępny dla innych procesów. Sprzęt powinien zapewnić ochronę pamięci operacyjnej, tzn. niemożność zaadresowania miejsca pamięci nienależącego do przydzielonego obszaru(ów). Jeśli architektura pamięci operacyjnej takich możliwości nie stwarza, będziemy musieli polegać na „dobrej woli" programistów, koniecznej dla wyjątkowo precyzyjnego testowania oprogramowania użytkowego (np. przekroczenie wartości wskaźnika w tablicy spowoduje zapis w obszarze kodu innego procesu i doprowadzi do jego błędnego działania).

Problem gospodarki pamięcią operacyjną możemy zracjonalizować przez następujące przedsięwzięcia:

- **priorytet pamięci operacyjnej**; proces, oprócz priorytetu określającego jego względną ważność przy dostępie do procesora, deklaruje (lub otrzymuje) dodatkowo priorytet dostępu do pamięci operacyjnej; technika ta ma sens w przypadku korzystania z pamięci zewnętrznej i dokonywania wymian między pamięcią dyskową a operacyjną;
- **niewywłaszczalność z pamięci operacyjnej**; proces jest opatrzony cechą lub przez zlecenie systemowe może zadeklarować na określony czas niewywłaszczalność z pamięci operacyjnej (analogiem jest *„sticky bit"* w systemie Xenix);
- **zapas sekcji do wynajęcia**; system kreuje zbiór obszarów pamięci operacyjnej o różnych rozmiarach (sekcji stałych) i w razie potrzeby wydzierżawia je procesom; sekcje są przy tym podzielone na klasy (np. małe, średnie, duże) w celu przyspieszenia odszukania obszaru o odpowiednim rozmiarze, a następnie ponownego skatalogowania po zwrocie lub rewindykacji. Sekcje takie mogą być zasobami niewywłaszczalnymi (uwaga na blokadę wzajemną!).

W przypadku braku pamięci zewnętrznej, podział pamięci w systemie między kod procesów wykonuje projektant systemu, natomiast w dalszym ciągu pozostaje problem gospodarowania pamięcią do przechowywania danych.

Wejście–wyjście

Wykonywanie operacji wejścia–wyjścia jest istotą systemu czasu rzeczywistego; ma on zareagować w ograniczonym czasie na zdarzenie pojawiające się w otoczeniu komputera. Proces użytkowy musi zatem otrzymać dane, na podstawie których właściwa reakcja zostanie wyznaczona i przesłana jako odpowiedź na to zdarzenie.

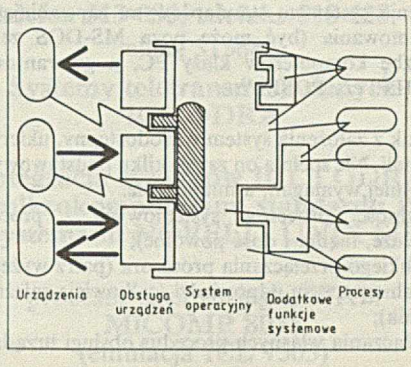
Urządzenia zewnętrzne, z których pobiera się dane i do których przesyła się wyniki różnią się często od „klasycznych" urządzeń peryferyjnych.

Urządzenie takie jest połączone z komputerem za pośrednictwem specjalizowanego sterownika. Sterownik „reprezentuje" to urządzenie, dopasowując protokół wymiany danych na poziomie fizycznym. Sterownik pracuje pod nadzorem właściwej dla niego procedury obsługi (ang. *driver*), realizującej dopasowanie protokołu wymiany na poziomie logicznym. Wymiana danych odbywa się zwykle z wykorzystaniem mechanizmu przerwań. W ten sposób sterownik zgłasza obecność danych do odebrania lub gotowość do ich wysłania.

Przerwanie pochodzące od sterownika urządzenia wejścia–wyjścia aktywizuje jądro, a ono z kolei musi uaktywnić „uśpiony" proces wykonujący kod drivera. Proces ten, po wykonaniu właściwych dla obsługi sterownika operacji (niektóre mogą być nierzerywalne), odblokuje proces użytkowy, odpowiadający za przetwarzanie danych w tym urządzeniu.

Częstość przerwań oraz zakres wykonywanych operacji w oczywisty sposób zależą od rodzaju urządzenia. Zależą również od „inteligencji" sterownika (czy może on buforować dane i jaki ma rozmiar bufora, jakie operacje wstępnego przetwarzania są wykonywane lokalnie w sterowniku itp.).

Z punktu widzenia oprogramowania użytkowego, operacje wejścia–wyjścia są wykonywane „gdzieś" w systemie operacyjnym, na ogół na poziomie niedostępnym dla procesów użytkowych. Równocześnie różnorodność urządzeń zewnętrznych i zastosowań wymusza taką konstrukcję systemu operacyjnego czasu rzeczywistego, aby dołączanie nowych urządzeń i związane z nim często programowanie właściwych driverów mógł samodzielnie wykonywać użytkownik.



Rys. 2.

System powinien być zatem otwarty i w prosty sposób rekonfigurowalny, być może nawet w sposób dynamiczny, tzn. w trakcie normalnej pracy. Powinien ponadto dostarczyć środki do konstruowania procedury obsługi, gdyż ta ostatnia staje się w pewnym sensie jego częścią i dlatego musi spełniać określone wymagania funkcjonalne.

Gospodarkę urządzeniami zewnętrznymi, procedurami ich obsługi oraz związanymi z tym strukturami danych realizuje warstwa systemu operacyjnego nazywana zwyczajowo monitorem wejścia–wyjścia.

System zbiorów

System zbiorów (metody logicznego dostępu do danych w pamięci masowej) nie jest standardowym składnikiem systemu operacyjnego czasu rzeczywistego. Jeśli ten poziom systemu operacyjnego występuje, to jego zadaniem jest konwersja operacji logicznych na danych w pamięci zewnętrznej na operacje fizyczne, nadzorowane przez monitor wejścia–wyjścia. Jego zadaniem jest również gospodarka pamięcią zewnętrzną: udostępnianie wolnych obszarów do zapisu danych, rewindykacja obszarów zwalnianych oraz eliminacja konfliktowych koincydencji w dostępie do danych (np. próba współbieżnego zapisu i odczytu przez różne procesy tego samego zespołu danych – rekordu, grupy rekordów, zbioru.

W systemie czasu rzeczywistego wynikiem krytycznym zwykle nie jest ochrona przed niepowołanym dostępem do danych, natomiast podstawowe znaczenie ma pewność przechowywania danych oraz opóźnienia w dostępie.

Pewność przechowywania danych polega na ochronie danych przed skutkami awarii, zarówno sprzętowych (np. brak zasilania), jak i programowych.

Opóźnienia w dostępie powstają na skutek narzutu wynikającego z konwersji logicznych adresów danych w ich fizyczne położenie na dysku (tylko tyle może „zrozumieć" sterownik dysku) oraz opóźnień transportu danych w wyniku ruchu głowic, fragmentacji danych (zapis w obszarze wielospójnym) lub „nieszczęśliwej" lokalizacji danych na odległym miejscu dysku.

Obydwa wymagania są wzajemnie sprzeczne, gdyż prewencja przeciwawaryjna w wyraźnym stopniu zwiększa narzut czasowy, bowiem systemu musi działać „ostrożnie", stale zabezpieczajc się przed możliwością wystąpienia awarii.

Oprogramowanie systemu zbiorów wprowadza ponadto znaczny narzut pamięciowy, powodując, że system operacyjny staje się „nieznośnie" duży. Dlatego rozwiązaniem dość często spotykanym jest przygotowanie w systemie czasu rzeczywistego środków do komunikacji z innym komputerem, w którym funkcjonuje system zbiorów. Obecnie, w tym innym komputerze z dużym prawdopodobieństwem spodziewać się można systemu operacyjnego typu UNIX.

A co z UNIXEM?

Spróbujmy teraz odpowiedzieć na postawione w tytule pytanie, dlaczego w funkcji systemu operacyjnego czasu rzeczywistego nie można użyć systemu typu UNIX.

UNIX jest systemem o długiej i dobrej tradycji. Jest praktycznie jedynym obecnie systemem standardowym, zapewniającym przenośność oprogramowania (być może poza MS-DOS ze względu na olbrzymią liczbę komputerów klasy PC, a gwarantowanym przez standardy SVID[2] czy POSIX[3].

Jest to jednak z założenia system wielodostępny, ukierunkowany na obsługę terminali. Nie spełnia on zatem kilku podstawowych, przedstawionych wcześniej wymagań, a mianowicie:
- wywłaszczalności procesów systemowych z procesora (jądro UNIX-a jest duże, mądre i dość powolne);
- bardzo szybkiego przełączania procesora (patrz wyżej);
- przewidywalnego czasu odpowiedzi (całkowicie zależny od chwilowego obciążenia);
- łatwego dołączania własnych procedur obsługi urządzeń zewnętrznych;
- szybkich mechanizmów komunikacji i wzajemnej synchronizacji procesów.

UNIX izoluje oprogramowanie użytkowe od sprzętu (i dlatego upraszcza jego przenośność) oraz izoluje między sobą procesy użytkowe (mechanizm *„mail"* to naprawdę nie jest to, o co chodzi w systemie czasu rzeczywistego). UNIX jest ponadto „dyskożerny" i bez pamięci zewnętrznej w ogóle nie może funkcjonować.

Mimo tego urok UNIX-a jest tak ogromny, że obecnie tworzone systemy czasu rzeczywistego bardzo często „coś" z niego mają i w znacznym stopniu na UNIX-ie bazują.

Wynika to z istnienia w środowisku UNIX-a bardzo obszernego oprogramowania narzędziowego, koniecznego przy produkcji oprogramowania użytkowego, a także z rozwiniętych mechanizmów komunikacji międzysystemowej.

Ważnym argumentem jest również i to, że programiści ze „stajni" UNIX-a są uważani powszechnie za dobrych specjalistów („jeśli się nauczył UNIX-a, to musi być zdolny!).

Przykłady

Dla ilustracji przedstawionych wyżej tez posłużę się trzema przykładami handlowo dostępnych systemów czasu rzeczywistego. W ich charakterystykach zwrócimy głównie uwagę na związki z systemem UNIX.

VxWorks

System operacyjny VxWorks jest produktem firmy Wind River System. Inc.[4]. Charakteryzuje się następującymi cechami:
- właściwości wymagane od systemu czasu rzeczywistego: wielozadaniowość z zastosowaniem priorytetów bezwzględnych, mechanizmy komunikacji i synchronizacji wzajemnej procesów, modyfikowalna obsługa przerwań, „strażnicy" czasowi, gospodarka pamięcią operacyjną;
- środki do pracy w sieci: „przezroczysty" dla procesorów użytkowych dostęp do innych systemów typu VxWorks i/lub UNIX za pośrednictwem mechanizmu „wtyczek" (ang. *socket*) UNIX-a i wynikające stąd zdalne wykonywanie zleceń (RPC[5]) oraz zdalny dostęp do danych; używa się protokołu sieciowego TCP/IP „nad" sprzętem Ethernet lub komunikacji za pośrednictwem magistrali w komputerze wieloprocesorowym;
- ładowalność oprogramowania: VxWork umożliwia ładowanie do pamięci modułów w standardowym formacie UNIX-a (OMF[6]), z dysku własnego lub za pośrednictwem sieci;,
- pośrednictwo: VxWorks jest wyposażony w pośrednictwo z użytkownikiem, podobne do „csh";
- system wejścia–wyjścia: zgodny na poziomie źródłowym ze standardem UNIX-a, wyposażony w drivery konsoli (tty), sieciowy, potokowy (ang. *pipe*), dysku RAM oraz standardowych urządzeń zewnętrznych (dysk, RS232);
- biblioteki użytkowe: komunikacja i synchronizacja procesów, obsługa buforów, kolejek i list, rejestracja zdarzeń, operacje na ciągach znaków itp.;
- oprogramowanie po skonfigurowaniu z systemem może być zapisane w pamięci ROM.

Zdaniem autorów tego systemu UNIX jest doskonałym systemem do produkcji oprogramowania i pracy interakcyjnej, natomiast systemy operacyjne czasu rzeczywistego tradycyjnie nie dostarczały żadnych narzędzi do budowy i produkcji oprogramowania, być może poza asemblerem i programem łączącym.

Z tego powodu postanowiono „zsieciować" VxWorks z UNIX-em. W czasie tworzenia oprogramowania VxWorks jest dostępny przez UNIX-a i pełni rolę „serwera czasu rzeczywistego". W normalnej pracy role odwracają się i UNIX pełni dla VxWorks (a ściślej dla wykonywanych w nim procesów użytkowych) funkcję serwera komunikacyjnego i udostępnia zbiory dyskowe.

Oprogramowanie użytkowe w systemie UNIX tworzy się za pomocą standardowych narzędzi, najlepiej w języku C. Moduły wynikowe są ładowane i łączone dynamicznie przez VxWorks i pod jego nadzorem wykonywane.

Bez systemu UNIX VxWorks może działać wtedy, gdy użytkownik posługując się UNIX-em doprowadzi VxWorks wraz z oprogramowaniem użytkowym do pełnej samodzielności.

VRTX32

System ten jest produktem firmy Ready Systems[7]. Jego charakterystyczną cechą jest bez wątpienia to, że sprzedawany jest w postaci pamięci PROM w wersjach dla następujących modeli procesorów:
- Motorola 68000/08/10/20/30;
- Intel 86/186/286/386;
- National Semiconductor seria 32000;
- AMD Am29000.

Uzupełnieniem systemu jest środowisko do tworzenia oprogramowania użytkowego, nazwane RTC (Real Time C), działające pod nadzorem systemów UNIX lub VMS (komputer VAX).

Dodatkową, mocną stroną VRTX32 jest dostarczany przez producenta zestaw narzędzi o nazwie CARDtools[8]. Twórcy wyszli przy tym z założenia, że narzędzia opracowane do projektowania oprogramowania w zastosowaniach „klasycznych" (nie czasu rzeczywistego) są tu nieadekwatne, gdyż nie pozwalają na:
– obsługę niestandardowych urządzeń zewnętrznych;
– integrację sprzętu i oprogramowania na poziomie projektu;

[2] UNIX System V Interface Definition – standard określający zestaw i zakres funkcji systemowych koniecznych w systemie operacyjnym zgodnym z AT & T UNIX System V.
[3] POSIX – Protable Operating System – podobny do UNIX-a (nazwę UNIX zastrzegł dla siebie koncern AT & T) standard określający pośrednictwo funkcjonalne, gwarantujące przenośność oprogramowania użytkowego, opracowany przez IEEE (USA).
[4] 1351 Ocean Avenue, Emeryville, California, USA.
[5] RPC – Remote Procedure Call – wykonywanie ciała procedur „gdzieś" w sieci.
[6] OMF – Object Module Format – standard zapisu modułów dających łączyć się w programy wynikowe, stosowany przez kompilatory języków programowania systemu UNIX.
[7] 470 Potrero Avenue, Sunnyvale, California, USA; Ready Systems SARL (Paris), 16 Bis Rue Grange Dame Rose, Velizy-Villacoublay, France
[8] CARD – Computer-Aided Real-Time Design

– projektowanie oprogramowania wykonywanego współbieżnie (niesekwencyjnie).

Większość systemów czasu rzeczywistego wymaga specyficznej (otwarte jądro) obsługi przerwań protokołów wymiany danych, narzucanych przez specjalizowane procesory peryferyjne i urządzenia zewnętrzne.

Chyba najciekawszym elementem CARDtools jest moduł służący do oceny wydajności oprogramowania w trybie czasu rzeczywistego – RTPV (Real-Time Performance Verification).

Zestaw CARDtools jest przygotowany do pracy na komputerach VAX, MicroVAX i VAXStation (system operacyjny VMS, UNIX lub ULTRIX) oraz komputerach SUN 3 i SUN 4 (system operacyjny SUNOS – implementacja systemu UNIX).

VENIX

System ten jest wspólnym produktem firm VenturCom Inc.[9] oraz INTERACTIVE Systems Corporation. Druga z wymienionych firm jest autorem systemu 386/ix – odpowiednika AT & T UNIX System V.3.2/386, a więc systemu klasy UNIX dla procesorów typu 386 (80386, 80386SX, 80486).

Wprowadzone przez VenturCom Inc. modyfikacje polegają na:
- ograniczeniu czasu przełączania kontekstu; dla komputera COMPAQ Deskpro 386 (16 MHz) czas ten jest nie dłuższy od 2 milisekund;
- priorytety bezwzględne; proces użytkowy o dużym priorytecie wywłaszcza również procesy systemowe („kernel" 386/ix);
- gwarantowany czas obsługi; ustala się procentowy udział procesu lub grupy procesów w wykorzystaniu czasu procesora, co chroni przed zdominowaniem procesora przez „nieodpowiedzialny" proces lub zadanie;
- bezpośredni dostęp do dowolnego miejsca pamięci operacyjnej i dowolnego portu we–wy, udostępniany procesom czasu rzeczywistego;
- w sytuacji, gdy proces czasu rzeczywistego wymaga komunikacji z urządzeniem wykorzystującym przerwania w transmisji danych, VENIX proponuje obsługę za pośrednictwem „minimalnego" drivera, od którego proces użytkowy może już bezpośrednio pobierać lub przekazywać dane (przez dostęp do jego buforów w pamięci operacyjnej);
- „zatrzaskiwanie" w pamięci operacyjnej; proces może zażądać nadania mu cechy niewywłaszczalności z pamięci operacyjnej celem skrócenia czasu odpowiedzi;
- wstępny podział pamięci operacyjnej; standardowo UNIX gospodaruje pamięcią operacyjną „biegu", dokonując na żądanie wymian z pamięcią dyskową. VENIX umożliwia wstępny podział pamięci, eliminując opóźnienia wprowadzane przez mechanizm gospodarki pamięcią operacyjną.

Z wymienionych powyżej systemów operacyjnych czasu rzeczywistego VENIX jest najbardziej UNIXopodobny – jest on po prostu zmodyfikowaną wersją systemu UNIX, w tym przypadku 386/ix.

* * *

Przedstawiona w artykule charakterystyka systemu operacyjnego czasu rzeczywistego miała na celu jedynie zasygnalizowanie najistotniejszych cech takiego systemu i jest, bez wątpienia, dość pobieżna. Jej celem było jednak wyłącznie wskazanie na wybrane ograniczenia i uwarunkowania wynikające z takiego właśnie trybu pracy i wykazanie, iż wymagań tych nie może spełnić system typu UNIX.

Zaprezentowane przykłady nie wyczerpują również różnorodności spotykanych systemów czasu rzeczywistego. Wybrano wyłącznie takich reprezentantów, którzy z systemem UNIX mają coś wspólnego.

[9] 215 First Street, Cambridge, MA, USA

Tylko prenumerata gwarantuje regularne otrzymywanie INFORMATYKI

Micomp

ZAKŁAD SYSTEMÓW MIKROKOMPUTEROWYCH

MiCOMP

40–045 Katowice, ul. Astrów 7
telefon 513-086, telefaks 518-628,
teleks 315687

Systemy teletransmisji danych ICL/ODRA

Integracja systemów ICL i ODRA z mikrokomputerami standardu PC i systemami NOVELL, UNIX SCO

procesor sieci rozległej WAN
MiCOMP 8075
(emulacja ICL 7503)

most NOVELL – ICL/ODRA
NetCROSS-FXBM
program teletransmisji danych
MiCROS-FXBM (PC, rozproszone bazy danych, wersja sieciowa – NOVELL),

skaner **MPXSCAN-8007,**

adaptery, modemy, testery, emulatory terminali

karty BSC do połączeń komputerów:
PC/XT/AT z PC/XT/AT
PC/XT/AT z **IBM**/370, R32, R34
z emulatorami IBM 3270/3780
PC/XT/AT z **ICL**

karty kompatybilne z **IRMA** do połączeń
PC/XT/AT z IBM 370
lokalnie przez IBM 3274

karty **SNA/SDLC**

konwertery o szybkościach do 19200 bps do zdalnego przyłączania terminali

przystawki teleksowe współpracujące z mikrokomputerami PC/XT/AT

Instalacje w największych systemach na terenie całego kraju!

0/23/90

POLSKIE
TOWARZYSTWO
INFORMATYCZNE

WOJCIECH TROJNIAR
Instytut Informatyki
Teoretycznej i Stosowanej PAN
Gliwice

Metody implementacji systemów czasu rzeczywistego

Celem niniejszego artykułu jest przedstawienie możliwie szerokiego spektrum dostępnych w naszym kraju systemów czasu rzeczywistego. Systemy te nie stanowią wprawdzie reprezentacyjnej grupy, jednak są zróżnicowane pod względem zakresu zastosowania i cech implementacyjnych. Omówione zostaną implementacje systemów QNX, XINU, RTM, MONTR i MCR-51, znaczenie pojęcia procesu (zadania) w tych systemach oraz sposób, w jaki procesy dzielą zasoby komputera.

Sposób impelemntacji ma duży wpływ na takie parametry systemu, jak czas reakcji systemu, ochrona zasobów, częstotliwość przełączeń oraz łatwość tworzenia aplikacji. Każdy z tych systemów jest ponadto przeznaczony do pracy w innym otoczeniu programowym. Szczególną uwagę poświęcono sposobom przełączania procesów, tzn. znalezieniu odpowiedzi na pytania: kiedy, w jaki sposób i które z nich zostaną przełączone.

Otoczenie systemów czasu rzeczywistego

W artykule ograniczono się do systemów implementowanych na mikroprocesorach INTEL-80 × 86/88. Najbardziej rozpowszechnionym oraz największym z tych systemów jest **system QNX** opracowany w kanadyjskiej firmie QUANTUM. Jest to dyskowy system operacyjny, który może zajmować oddzielną partycję na dysku stałym. Mikrokomputery pracujące pod nadzorem tego systemu można łączyć w sieć działającą według protokołu ARCNET. Użytkownik, który pierwszy raz widzi ten system, może odnieść wrażenie, że jest to jeszcze jedna implementacja systemu UNIX. Jednak po bardziej uważnej analizie dojdzie on do wniosku, że jest to system czasu rzeczywistego, w którym zachowano unixowy styl programowania.

System XINU został wykonany na Uniwersytecie Purdue (USA). Tekst programu oraz omówienie systemu zostały wydane w postaci książki. Zakres zastosowania systemu jest dość szeroki, został bowiem zaprojektowany w sposób warstwowy. Użytkownik może wykorzystywać podzbiór funkcji systemu operacyjnego i w ten sposób, w zależności od aplikacji, XINU można stosować np. w jednokartowych mikrokomputerach lub jako dyskowy system operacyjny połączony w sieć. System XINU jest pozbawiony procesora poleceń. Użytkownik musi wygenerować cały system operacyjny na komputerze, wyposażonym w system uruchomieniowy. Tak wygenerowany XINU może zostać zaprogramowany w pamięci stałej lub wpisany na dysk maszyny docelowej. Początkowo XINU został opracowany dla systemów wyposażonych w mikroprocesor LSI-11. Dla mikroprocesora INTEL 8086 opracowano oprogramowanie uruchomieniowe na minikomputerze VAX pod nadzorem systemu Unix. System ten stał się podstawą do zrealizowania oprogramowania uruchomieniowego systemu XINU dla kompilatora C firmy Microsoft. Z punktu widzenia użytkownika, XINU stanowi bibliotekę, której procedury tworzą jądro systemu dołączane do programu użytkownika. Połączone oprogramowanie można załadować do pamięci systemu docelowego lub zaprogramować w pamięci stałej.

System RTM, zrealizowany w firmie POLGAT w Gliwicach, został pomyślany jako rezydentny moduł systemu DOS na mikrokomputerze klasy IBM. Oprócz podstawowych funkcji systemu wielozadaniowego RTM zawiera również pewne funkcje ochrony systemu DOS przed próbą wykonania funkcji systemowych przez kilka procesów. Ochrona taka jest niezbędna, ponieważ procedury systemu DOS nie są wielowejściowe. System RTM znalazł zastosowanie w wielu aplikacjach, w których zostały użyte sterowniki Simatic S5.

Monitor wielozadaniowy MONTER został wykonany na zlecenie PTI. Jest on samodzielnym modułem programowanym dołączonym w całości do programu użytkownika. Został napisany w języku C i skompilowany kompilatorem firmy Microsoft. Monitor może pracować w środowisku systemu DOS lub być częścią programu umieszczonego w pamięci stałej systemu.

Powyższy przegląd środowisk, w jakich mogą być implementowane systemy czasu rzeczywistego, wykazuje ich duże zróżnicowanie. W jeszcze większym stopniu są one zróżnicowane, jeżeli weźmiemy pod uwagę szczegóły implementacyjne systemów.

Procesy

Przedmiotem rozważań artykułu są jednoprocesowe systemy czasu rzeczywistego. Pojęcia procesu dla każdego z tych systemów różnią się między sobą. Poniżej zostanie przedstawiony sposób impelementacji procesów w wybranych systemach.

Proces wykonywany przez procesor może zostać przerwany na skutek zewnętrznego zdarzenia. Aby przerwać proces, należy zachować stan procesu w określonym miejscu pamięci zwanym „zmiennym otoczeniem" (ang. volatille enviroment). Takie otoczenie składa się zwykle ze stałego obszaru w pamięci danych, zwanego deskryptorem procesu, oraz z fragmentu obszaru stosu, w którym jest przechowywany stan rejestrów. W niektórych implementacjach stan rejestrów jest przechowywany w deskryptorze procesów. Struktura deskryptora procesu jest zależna od otoczenia, w jakim proces jest wykonywany, mechanizmów synchronizacji i komunikacji, zasobów administrowanych przez proces, rodzajów procesów przewidzianych w systemie itp.

W systemie QNX proces (zadanie) jest tworzony z programu zwanego SHELL lub z innego procesu. Proces jest tworzony na podstawie pliku programu. Każdy proces ma zasoby pamięci oraz konsolę oznaczaną w tym systemie przez TTY. Proces może odwoływać się bezpośrednio do tych danych, które sam utworzył. Procesy użytkownika nie mogą tworzyć danych globalnych, tzn. danych bezpośrednio dostępnych przez kilka procesów jednocześnie. Jedyną możliwością wymiany informacji między procesami jest przesyłanie komunikatów między procesami. Komunikat z jednego procesu do innego jest wysyłany przez przepisywanie obszaru pamięci, a nie przez przesłanie adresu obszaru pamięci, w którym znajduje się treść komunikatu. Wynika to z przyjętej filozofii sieciowego systemu operacyjnego. Proces może przesyłać komunikaty do procesu znajdującego się w innej maszynie w podobny sposób jak do procesu, znajdującego się w tej samej maszynie. Innym elementem określającym stan procesu, jest usytuowanie go pośród innych procesów oraz mechanizm ochrony danych. Ponieważ QNX jest pełnym dyskowym systeme operacyjnym, wzorowanym na systemie UNIX, mechanizmy ochrony mają swoje źródło również w tym systemie, aczkolwiek nie są takie same. Podobnie jak w systemie UNIX, wprowadzono pojęcie użytkownika oraz grupy użytkowników. W miejsce unixowego pojęcia sygnału wprowadzono pojęcie wyjątku (ang. exception). Pojęcia powyższe mają swoje odzwierciedlenie w deskryptorach procesów. Na wydruku 1 przedstawiono skomentowany (częściowo) wydruk deskryptora procesu systemu QNX.

Wydruk 1 zawiera jedynie część deskryptora procesu umieszczonego w pamięci danych systemu, natomiast jego pozostała część znajduje się w pamięci procesu. W systemie QNX przeznaczono do tego celu początkowe 128 bajtów pamięci danych procesu. Struktura tego obszaru pamięci jest pokazana na wydruku 2.

```
struct_task_entry {
struct_task_entry * tlink;              /*łącznik w kolejce procesów*/
unsigned tid;                           /*numer identyfikacyjny procesu*/
unsigned blocked_on;
char * send_msg;                        /*adres przesyłanego komunikatu */
unsigned msg_length                     /*długość komunikatu*/
unsigned data_seg;                      /*segment danych*/
unsigned data_size;                     /*rozmiar danych*/
char state;                             /*stan procesu*/
char priority;                          /*priorytet procesu*/
char user_num;                          /*numer użytkownika*/
char task_tty;                          /*numer tty procesu*/
char group_num;                         /*numer grupy*/
char code_index;
unsigned flags;                         /*flagi procesu*/
struct task_entry sender_queue;         /*kolejka procesów oczekujących
                                        na potwierdzenie komunikatu*/
char * reply_msg;                       /*adres komunikatu potwierdze-
                                        nia*/
struct sys_mem_entry * segment list;/*lista obszarów pamięci zajętych
                                        przez proces*/
struct task_entry * father;             /*proces „ojciec"*/
struct task_entry*son;                  /*proces „syn"*/
struct task_entry * brother;            /*proces „brat"*/
unsigned task_tty_nid;                  /*numer węzła do którego jest
                                        dołączony tty*/
unsigned cd_disk_id;                    /*id. dysku b.k.*/
unsigned cd_nid;                        /*numer węzła b.k.*/
unsigned cd_block;                      /*numer bieżącego bloku dysk.*/
unsigned cd_index;                      /*index w b.k.*/
unsigned ext_cd_block                   /*numer rozszerzenia*/
char cd_drive;                          /*numer dysku*/
char team_num;
```

Wydruk 1. Częściowo skomentowany wydruk deskryptora procesu systemu QNX; skrót b.k. oznacza „bieżąca kartoteka"

```
unsigned User_sp;                       /*rejestr SP*/
unsigned User_ss;                       /*rejestr SS*/
unsigned Exc_bits [2];                  /*Wyjątek istnieje*/
unsigned Exc_allow [2];                 /*Wyjątek dozwolony*/
unsigned Exc_pending [2];               /*Aktywny wyjątek*/
unsigned Exc_fp;                        /*adres wyjątku*/
unsigned Exc_pc;                        /*adres funkcji wyjątku*/
unsigned Exc_permit;                    /*Wyjątek dozwolony*/
int Exit_status;                        /*Status wyjścia*/
int Error_status;                       /*Status błędu*/
unsigned My_tid;                        /*id procesu*/
unsigned Dad_tid;                       /*id procesu ojca*/
char Brek_count;                        /*poziom przerwania procesu*/
char My_tty                             /*TTY procesu*/
char Cmd_flags;                         /*Flagi procesu*/
char My_priority;                       /*Priorytet procesu*/
FILE stdin;                             /*Standardowe wejście*/
FILE stdout;                            /*Standardowe wyjście*/
FILE stderr;                            /*Standardowe urz. błędu*/
char *Free_list;                        /*Adres początku wolnej pamięci*/
char *End_of_free;                      /*Adres końca wolnej pamięci*/
FILE *File_list;                        /*Adres listy otwartych plików*/
int _Argc;                              /*Liczba argumentów w linii ko-
                                        mendy*/
char *_Argv [];                         /*Adres tablicy adresów tekstu linii
                                        komendy*/
char Code_flags;                        /*Flagi komendy*/
char User_num;                          /*Numer użytkownika*/
char Group_num;                         /*Numer grupy*/
char Team_num;
unsigned My_nid;                        /*Numer węzła*/
unsigned Dads_nid;                      /*Numer węzła procesu ojca*/
unsigned My_tty_nid;                    /*Numer węzła na którym jest TTY
                                        procesu*/
unsigned My_cd_nid;                     /*Numer węzła w którym znajduje
                                        się kartoteka procesu*/
char Release_num;                       /*Numer wersji*/
unsigned * Segment_table*ptr;           /*Adres tabeli adresów segmentów
                                        dla dużego modelu*/
unsigned Qdb_tid;                       /*id debugger'a QDB*/
```

Wydruk 2. Struktura pamięci danych procesu w systemie QNX

Dla systemów, które są tylko jądrem czasu rzeczywistego, deskryptory procesów są znacznie mniej skomplikowane. Podstawowymi funkcjami jądra systemu jest umożliwienie tworzenia procesów, wzajemna komunikacja i synchronizacja procesów, zarządzanie pamięcią operacyjną oraz zapewnienie reakcji systemu na zdarzenia zewnętrzne, w tym przypadku przerwania od zegara i urządzeń. Ponieważ otoczenie systemowe jest mniej skomplikowane niż w systemie QNX (XINU jest tylko bliblioteką wielozadaniową), zachowanie i odtworzenie stanu procesu wymaga mniej zabiegów niż w przypadku procesu wykonywanego pod nadzorem systemu QNX. Deskryptor procesu XINU, pokazany w wydruku 3, ma następującą postać:

```
struct pentry {
char   pstate;          /* stan procesu */
short  pprio;           /* priorytet procesu */
short  pregs [PNREGS];  /* rejestry procesu */
short  psem;            /* semafor na którym oczekuje proces */
short  pmsg;            /* komunikat przesłany do procesu */
short  phasmsg;         /* ! = 0, jeżeli istnieje komunikat */
short  pbase;           /* baza stosu */
short  pstklen;         /* długość stosu */
short  plimit;          /* limit stosu */
char   pname [PNMLEN];  /* nazwa procesu */
short  pargs;           /* początkowa liczba argumentów */
short  paddr;           /* adres początku procesu */
};
```

Wydruk 3. Deskryptor procesu w systemie XINU

W systemie XINU, podobnie jak w pozostałych omawianych systemach, istnieje z góry określona liczba deskryptorów procesów, a co za tym idzie, liczba procesów, kótre mogą być wykonywane współbieżnie. Każdy proces ma swój stos, który jest deklarowany w czasie tworzenia procesu. Najczęstszym błędem programistów systemowych jest zarezerwowanie zbyt małego obszaru stosu dla danego procesu. Stosy wszystkich procesów znajdują się w jednym segmencie. Wykonanie operacji wymagających dużego obszaru stosu (np. formatowanie WE–WY) może spowodować zniszczenie obszaru stosu w innym segmencie. W celu wykrywania takich sytuacji przewidziano w deskryptorze specjalne pola, w których zapamiętano liczbę stosów oraz położenie stosu procesu w pamięci operacyjnej. Omawiając strukturę tego deskryptora, należy zwrócić uwagę na to, że stan rejestrów jest przechowywany w deskryptorze procesu, a nie na stosie.

Deskryptor systemu RTM przechowuje rejestry na stosie procesu. W deskryptorze znajdują się tylko rejestry *SS* i *SP*. System MONTR ma deskryptor podobny jak w systemie XINU i dlatego nie będzie tu omawiany.

Pytanie podstawowe zadane we wstępie, *w jaki sposób procesy można przełączać?*, znajduje tu odpowiedź. Stan aktualnie wykonywanego procesu należy zapamiętać w zmiennym otoczeniu jednego procesu, a następnie odtworzyć stan innego procesu i kontynuować wykonanie obu tych procesów. Nasuwa się pytanie, kiedy proces zostanie wywłaszczony i który proces będzie kontynuował wykonanie?

Przerwanie wykonania procesu

Proces może przerwać swoje wykonanie na skutek zdarzenia zewnętrznego lub gdy proces osiągnie stan, w którym dalsze wykonanie jest niemożliwe. Zdarzeniem zewnętrznym, które powoduje przerwanie wykonania procesu, jest przerwanie urządzeniowe. W tym momencie rozpoczyna się wykonanie procedury obsługi przerwania. Procedury obsługi można podzielić na dwa rodzaje:

- przerwanie przezroczyste dla systemu,
- przerwanie powodujące zmianę stanu systemu.

Z pierwszym przypadkiem mamy do czynienia wtedy, gdy np. zegar aktualizuje swój czas, lub gdy nadchodzi kolejny, wpisywany do bufora

znak z łącza szeregowego. Drugi przypadek jest wtedy, gdy np. proces oczekuje na skompletowanie linii z terminala i nadszedł właśnie znak *CR*, albo gdy proces został zawieszony w oczekiwaniu na upłynięcie czasu (czas upływa w momencie przerwania pochodzącego od zegara). Przerwaniami pierwszego rodzaju nie będziemy się dalej zajmować. W drugim przypadku reakcje omawianych systemów są zróżnicowane. Najprostszy mechanizm zastosowano w systemie QNX. Program obsługi przerwań jest tu podzielony na część programu użytkownika oraz część systemową. W części użytkownika musi być zachowany na stosie stan rejestrów procesora, potwierdzenie przerwania (*EOI*), oraz wywołanie procedury systemowej (*INT* 71), który sygnalizuje tzw. port. Port jest elementem systemu, w którym można dokonać operacji synchronizacji *signal, avait* i *receive*. Operacja *signal* może być wykonana w porcie albo przez procedurę programową, albo przez procedurę obsługi przerwań. Proces zawieszony w porcie przez wykonanie operacji *avait* lub *receive* może zostać odwieszony przez operację *signal*. Operacja *int* 71 jest wielowejściowa, w której tylko przez krótkie odcinki czasu przerwanie jest całkowicie zablokowane. Jeżeli w czasie wykonywania operacji *int* 71 wystąpi kolejne przerwanie wykonujące tę samą operację, to zostanie ono ustawione w kolejkę. Kolejka zgłoszeń jest sprawdzana przed zakończeniem operacji i obsługiwana aż do wyczerpania się wszystkich zgłoszeń. Dopiero po tym czasie są wykonywane inne funkcje systemu. Przerwane funkcje systemowe są wykonywane dopiero po obsłużeniu zgłoszeń pochodzących od przerwań. W tym systemie nie jest wyróżniony jakiś szczególny plan obsługi urządzeń. Istnieje program zarządzania niektórymi urządzeniami, takimi jak dysk, konsola lub terminale. Istnieje specjalna biblioteka umożliwiająca dołączanie nowych urządzeń jako plików, podobnie jak w systemie UNIX. Oprócz tego jest możliwe wykonywanie instrukcji WE–WY, zarządzania sterownikiem przerwań oraz procedura obsługi przerwań i zegara przez programy użytkownika.

W systemie RTM program procedury obsługi przerwania może być wykonywany w stanach *DI, I* oraz *S*. Stan *DI* jest to stan bezpośredniego wykonania przerwania w sposób przezroczysty dla systemu przy zablokowanym systemie przerwań. Ze stanu *DI* można przejść w stany *I* oraz *S* przez wywołanie funkcji systemu. W stanie *I* przerwania są zablokowane na poziomach równym i niższym niż aktualnie wykonywane. W stosie jest zachowane również część rejestrów. W tym stanie przerwanie o wyższym priorytecie może przerwać aktualną obsługę przerwania. Ze stanu *I* można przejść w stan *S*. Procedury umożliwiające przejście ze stanu *DI* oraz *I* do stanu *S* zachowują w stosie stan rejestrów procesora, przesyłają komunikat do programu zarządzającego zgłoszeniami oraz odblokowują system przerwań. Przed przejściem do stanu *S* powinny być obsłużone wszystkie przerwania w stanie *I*. Obsługa zgłoszenia powodująca przejście w stan *S* powoduje kontynuację wykonania programu obsługi przerwania od miejsca, w którym nastąpiło wywołanie przejścia w stan *S*. Po wykonaniu procedury obsługi w stanie *S*, program nie wraca do przerwanego procesu, lecz do programu obsługi zgłoszeń systemu. Program obsługi zgłoszeń pobiera z kolejki zgłoszenia pochodzące od przerwań i wywołań funkcji systemowych. Po zakończeniu obsługi następuje wykonanie wybranego procesu użytkownika. W systemie tym można deklarować urządzenia oraz drivery obsługi urządzeń. Wymaga to zdyscyplinowanego oprogramowania, niemniej czasy reakcji na przerwanie są dość krótkie.

O ile w systemach QNX i RTM można mówić o pewnych podobieństwach w traktowaniu zgłoszeń (w obu przypadkach zgłoszenia są przed obsługą ustawiane w kolejkę), to w systemie XINU obsługa odbywa się na innej zasadzie. Ceną za skrócenie czasu trwania odcinków nieprzerywalnych jest wyróżnienie określonych funkcji przeznaczonych jedynie do obsługi przerwań. W systemie XINU program obsługi przerwania może wywołać dowolną funkcję systemową. Program obsługi jest wykonywany przy zablokowanym systemie przerwań na koszt aktualnie przerwanego procesu. Nie istnieje również program obsługi zgłoszeń systemowych. Każda funkcja systemowa jest wykonywana jako procedura wywołującego ją procesu. Funkcje systemowe muszą zapamiętać w stosie aktualny stan przerzutnika blokady przerwania i odtworzyć ten stan po wykonaniu funkcji. Dobrą praktyką programowania w tym systemie jest wstawienie na końcu procedury obsługi przerwania oraz wywołania procedury sygnalizacji semafora licznikowego, który uprzednio został utworzony z wartością początkową 0. Przerwania w systemie XINU są przyporządkowane urządzeniom logicznym. Każde z uderzeń dysponuje procedurą obsługi przerwania pochodzącego od urządzenia wejściowego i wyjściowego (np. TTY jest jednocześnie urządzeniem wejściowym i wyjściowym). Ponadto przewidziano obsługę przerwań pochodzących od urządzeń, które nie zostały uwzględnione w strukturze urządzeń logicznych, nie jest wspomagana przez system.

System MONTR, podobnie jak system XINU, umożliwia wykonanie dowolnej funkcji systemowej z procedury obsługi przerwania. Dołączenie procedury obsługi przerwania do wektora przerwań jest wspomagane przez system. W systemie tym zrezygnowano z pojęcia urządzenia logicznego.

Dotychczas daliśmy odpowiedź na pytanie: *w jaki sposób i kiedy proces zostanie wywłaszczony?* Pozostaje pytanie: *który proces zostanie wybrany do wykonania?*

Szeregowanie procesów

Szeregowanie procesów polega na wyborze do wykonania procesu o największym znaczeniu. W systemach czasu rzeczywistego najczęściej rozważanym parametrem jest czas reakcji na zdarzenie zewnętrzne. Maksymalny czas reakcji powinien być możliwy do określenia. Stopień wykorzystania urządzeń oraz przepustowość systemu odgrywają tu rolę drugorzędną. Procesom przyporządkowuje się priorytety. Na podstawie wartości priorytetu dokonuje się wyboru procesu, który ma być kontynuowany. W systemach czasu rzeczywistego priorytety są deklarowane przez użytwkoników w momencie kreacji procesów. Stosuje się tu zwykle algorytm z wywłaszczeniem (ang. *preemptive*). Polega on na tym, że w momencie wystąpienia zdarzenia zmianiającego stan systemu (np. proces o wyższym priorytecie staje się aktywny z powodu otrzymania komunikatu lub sygnału od przerwania) jest wybierany proces o najwyższym priorytecie. Zmiana priorytetu jest również możliwa w systemach, a także odsunięcie w czasie zdarzeń powodujących zmianę stanu systemu. Taki mechanizm użyto w systemie XINU przy oprogramowaniu sieci. W tym przypadku w czasie transmisji bloku danych z prędkością 9600 bodów zmieniono priorytet procesu tracenia czasu, który zwykle ma najniższy priorytet, na priorytet najwyższy. W ten sposób żaden proces, oprócz procesu tracenia czasu nie może być wykonany przez procesor. Zdarzenia pochodzące z zegara systemowego są zapamiętywane, ale nie obsługiwane. Po zakończeniu transmisji, priorytet procesu tracenia czasu zmienia się na najniższy. Jest to sposób uniknięcia podczas transmisji bloków długich nieprzerywalnych odcinków czasu.

Monitor czasu rzeczywistego dla mikrokontrolera 8051

Monitor Czasu Rzeczywistego procesorów rodziny 8051 (MCR-51) jest zbudowany z wykorzystaniem właściwości architektury jednoukładowego procesora 8051. Głównymi założeniami funkcjonalnymi struktury MCR-51 jest maksymalizacja czasów reakcji na przerwania, minimalizacja czasów obsługi zgłoszeń do monitora przez procesy oraz minimalizacja obszaru pamięci. Możliwości funkcjonalne monitora są dostosowane do typowych zastosowań mikroprocesorów wspomnianej rodziny.

Podstawowym elementem monitora jest proces. Każdy proces jest to procedura, dysponująca pełnym zestawem rejestrów procesora, stosem o rozmiarze 16 bajtów oraz dowolną wielkością pamięci zewnętrznej. Każdy proces ma priorytet od 0 do 3, które są niezmienne. Proces może znajdować się w dwóch podstawowych stanach:

- gotowy do realizacji:
- oczekujący na zdarzenie (wolny semafor, przerwanie lub upływ oznaczonego czasu).

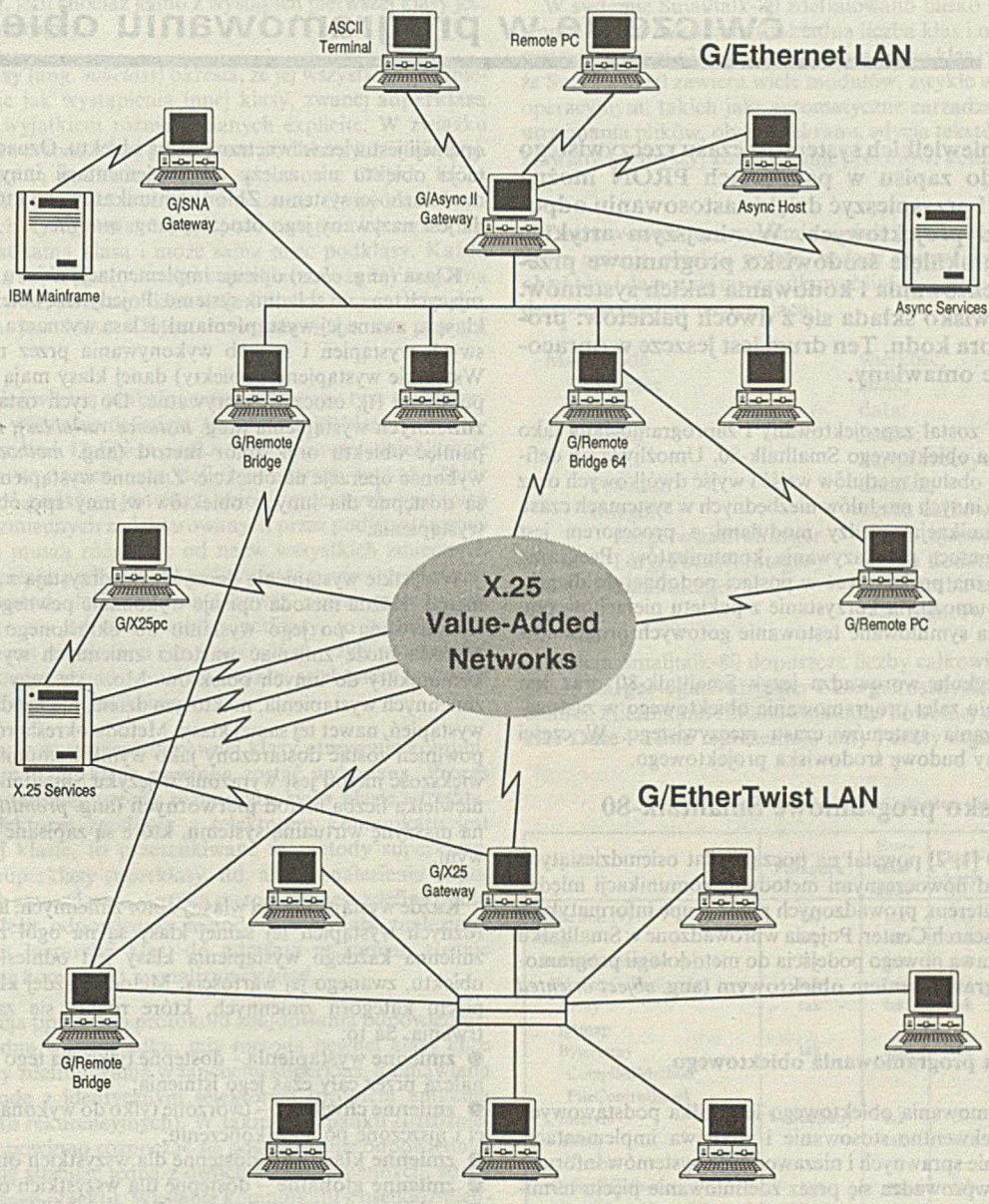
W danej chwili jest realizowany proces *„gotowy do realizacji"* o najwyższym priorytecie oraz mogą istnieć co najwyżej cztery procesy o różnych priorytetach. Możliwe jest dynamiczne tworzenie i usuwanie procesów. Przy zapoczątkowaniu działania monitora jest tworzony proces użytkownika o nazwie procedury *MAIN* i priorytecie 3. Dany proces może usunąć każdy inny i to niezależnie od priorytetu, a także siebie.

Kolejnym elementem monitora są semafory binarne. Jest ich 32 i po inicjacji wszystkie są w stanie *„wolny"*. Każdy semafor może być zajmowany, zwalniany lub testowany. Semafory są zorganizowane na mapie bitów i nie mają kolejek. Po zwolnieniu semafora, zawsze jest aktywizowany proces o priorytecie najwyższym.

dokończenie na s. 28

Produkty sieciowe LAN i WAN

Gateway
communications, inc.



BEZKONKURENCYJNE PRODUKTY
SIECI KOMPUTEROWYCH NAGRADZANE PRZEZ:
PC MAGAZINE, LAN MAGAZINE, INFO WORLD
OFERUJE AUTORYZOWANY DYSTRYBUTOR:

MIKROB

P.W.P.T. MIKROB SP. Z O.O.
20-601 LUBLIN, UL. ZANA 38 BLOK C, VIII p.
TEL. (0-81) 55-90-19, **FAX** 55-91-69, **TLX** 643776 MIKRO PL

Na życzenie wysyłamy katalog produktów. Atrakcyjny program współpracy dla dealer'ów.

POLSKIE
TOWARZYSTWO
INFORMATYCZNE

PAWEŁ GIZBERT-STUDNICKI
JERZY KARCZMARCZUK
Instytut Fizyki
Uniwersytet Jagielloński
Kraków

Środowisko do projektowania prostych systemów czasu rzeczywistego – ćwiczenie w programowaniu obiektowym

Projektowanie niewielkich systemów czasu rzeczywistego przeznaczonych do zapisu w pamięciach PRON można znacznie uprościć i przyspieszyć dzięki zastosowaniu odpowiednich narzędzi projektowych. W ninejszym artykule przedstawiamy zamknięte środowisko programowe przeznaczone do projektowania i kodowania takich systemów. Opisywane środowisko składa się z dwóch pakietów: projektanta i generatora kodu. Ten drugi jest jeszcze w opracowaniu i nie będzie omawiany.

Pakiet projektanta został zaprojektowany i zaprogramowany jako podśrodowisko języka obiektowego Smalltalk-80. Umożliwia on definiowanie programów obsługi modułów wejść i wyjść dwójkowych oraz analogowych, a także innych modułów niezbędnych w systemach czasu rzeczywistego. Komunikacja między modułami a procesorem jest zorganizowana za pomocą przekazywania komunikatów. Programy obsługi modułów można projektować w postaci podobnej do diagramów blokowych, co umożliwia korzystanie z pakietu niefachowcom. Pakiet pozwala też na symulowane testowanie gotowych programów obsługi.

Pierwsza część artykułu wprowadza język Smalltalk-80 oraz jest poświęcona omówieniu zalet programowania obiektowego w zastosowaniu do projektowania systemów czasu rzeczywistego. W części drugiej przedstawiamy budowę środowiska projektowego.

Język i środowisko programowe Smalltalk-80

Język Smalltalk-80 [1, 2] powstał na początku lat osiemdziesiątych jako owoc badań nad nowoczesnymi metodami komunikacji między człowiekiem a komputerem, prowadzonych przez grupę informatyków z Xerox Palo Alto Research Center. Pojęcia wprowadzone w Smalltalku szybko stały się podstawą nowego podejścia do metodologii programowania, zwanego **programowaniem obiektowym** (ang. *object oriented programming*).

Podstawowe pojęcia programowania obiektowego

U podstaw programowania obiektowego leży kilka podstawowych pojęć, których konsekwentne stosowanie i właściwa implementacja pozwalają na tworzenie sprawnych i niezawodnych systemów informatycznych. Pojęcia te wprowadza się przez zdefiniowanie pięciu terminów: **obiektu, komunikatu, klasy, wystąpienia** i **metody**. Niestety, definicja podanych terminów jest zapętlona, tj. każdy z nich jest definiowany za pomocą pozostałych.

Obiekt (ang. *object*) składa się pewnej liczby jednostek pamięci i pewnego zbioru operacji. Operacje obiektu zależą od tego, co obiekt reprezentuje. Obiekty reprezentują poszczególne składniki systemu m.in. liczby, łańcuchy, słowniki, procesy obliczeniowe, edytory tekstu itd.

Komunikat (ang. *message*) jest zleceniem wykonania przez obiekt pewnej operacji. Komunikat określa jaka operacja ma być wykonana, ale nie precyzuje w jaki sposób ją wykonać. O tym decyduje adresat (ang. *receiver*) – obiekt otrzymujący komunikat. Sposób wykonania operacji jest więc wewnętrzną cechą obiektu. Oznacza to, że implementacja obiektu nie zależy od implementacji innych obiektów, czyli modularności systemu. Zbiór komunikatów, na które obiekt odpowiada jest nazywany jego **otoczką** (ang. *interface*).

Klasa (ang. *class*) opisuje implementację zbioru obiektów reprezentujących ten sam składnik systemu. Pojedyncze obiekty opisywane przez klasę są zwane jej **wystąpieniami**. Klasa wyznacza organizację pamięci swych wystąpień i sposób wykonywania przez nie zbioru operacji. Wszystkie wystąpienia (obiekty) danej klasy mają podobne własności publiczne (tj. otoczkę) i prywatne. Do tych ostatnich należą: zbiór **zmiennych** wystąpienia (ang. *instance variables*), z których składa się pamięć obiektu oraz zbiór **metod** (ang. *methods*) opisujących jak wykonać operacje na obiekcie. Zmienne wystąpienia i jego metody nie są dostępne dla innych obiektów w inny sposób niż przez otoczkę wystąpienia.

Wszystkie wystąpienia danej klasy korzystają z tego samego zbioru metod. Każda metoda opisuje wykonanie pewnego komunikatu i jest wykonywana po jego wysłaniu do określonego wystąpienia klasy. Metoda może zmieniać wartości zmiennych wystąpienia i wysyłać komunikaty do innych obiektów. Może się przy tym odwoływać do zmiennych wystąpienia, na którym działa, ale nie do zmiennych innych wystąpień, nawet tej samej klasy. Metoda określa również obiekt, który powinien zostać dostarczony jako wynik komunikatu. Przytłaczająca większość metod jest wyrażona w języku Smalltalk-80. Istnieje jednak niewielka liczba metod **pierwotnych** (ang. *promitive*) składających się na maszynę wirtualną systemu, które są zapisane w kodzie maszynowym.

Każde wystąpienie ma własny zbiór zmiennych, ale zbiory zmiennych różnych wystąpień tej samej klasy są na ogół równoliczne. Każda zmienna każdego wystąpienia klasy jest odniesieniem do pewnego obiektu, zwanego jej wartością. Metody każdej klasy mają dostęp do pięciu kategorii zmiennych, które różnią się zakresami i czasami trwania. Są to:

- **zmienne wystąpienia** – dostępne tylko dla tego obiektu, do którego należą przez cały czas jego istnienia;
- **zmienne chwilowe** – tworzone tylko do wykonania pewnej czynności i niszczone po jej zakończeniu;
- **zmienne klasowe** – dostępne dla wszystkich obiektów danej klasy;
- **zmienne globalne** – dostępne dla wszystkich obiektów;
- **zmienne grupowe** – dostępne dla wszystkich obiektów pewnej grupy klas.

Zmienne wystąpienia dzielą się dalej na dwie kategorie: **zmienne nazwane** (ang. *named instance variables*) oraz **zmienne indeksowane** (ang. *indexed instance variables*). Do pierwszej kategorii należą te zmienne, do których metody odnoszą się przez podanie identyfikatora, do drugiej zaś zmienne, do których wszystkie odniesienia następują za pomocą specjalnych komunikatów z indeksami całkowitymi. Każdy obiekt danej klasy ma tę samą liczbę zmiennych nazwanych, natomiast dwa obiekty tej samej klasy mogą mieć różną liczbę zmiennych indeksowanych. Istnienie zmiennych indeksowanych pozwala na łatwe definiowanie w Smalltalku sekwencyjnych struktur danych. Wraz z nim są tworzone wszystkie zmienne wystąpienia obiektu.

Hierarchia klasowa

W systemie programowania obiektowego powinny istnieć środki ułatwiające oddziaływania między definicjami różnych klas. Inaczej trudno byłoby opisać np. klasy liczb całkowitych i rzeczywistych, które zachowują się podobnie odpowiadając na komunikaty operacji arytmetycznych, ale inaczej reprezentują liczby. Najogólniejszym sposobem wprowadzenia związków między różnymi klasami jest tzw. **wielorakie dziedziczenie** (ang. *multiple inheritance*), w którym pewne obiekty mogą być wystąpieniami więcej niż jednej klasy na raz. W systemie Smalltalk-80 wybrano jednak mniej ogólną postać związków międzyklasowych, wziętą z języka Simula i zwaną **podziałem na podklasy** (ang. *subclassing*). Podział na podklasy jest ściśle hierarchiczny, tj. wymaga by wszystkie wystąpienia pewnej klasy były równocześnie wystąpieniami innej, jeśli chociaż jedno z wystąpień pierwszej klasy jest także wystąpieniem drugiej.

Definicja **podklasy** (ang. *subclass*) określa, że jej wszystkie wystąpienia będą takie same jak wystąpienia innej klasy, zwanej **superklasą** (ang. *superclass*) z wyjątkiem różnic podanych explicite. W związku z tym każda nowo tworzona klasa jest zawsze podklasą już istniejącej. Wszystkie inne klasy są podklasami klasy systemowej **Object**, która opisuje wspólne cechy wszystkich obiektów systemu. Opisy klas określają, w jaki sposób ich wystąpienia różnią się od wystąpień ich superklas. Każda podklasa jest sama klasą i może sama mieć podklasy. Każda klasa ma tylko jedną superklasę, chociaż superklasa może być wspólna dla wielu klas. Tak więc, struktura klas w Smalltalku-80 jest drzewem, którego korzeniem jest klasa **Object** – jedyna klasa nie mająca superklasy.

Podklasa musi mieć własną unikalną nazwę ale, poza tym dziedziczy po swej superklasie wszystkie definicje, tj. zarówno deklaracje zmiennych, jak i definicje metod. Definicja podklasy może zawierać deklaracje nowych zmiennych wystąpień oraz definicje nowych metod. Zbiór zmiennych wystąpień podklasy jest sumą zbioru zmiennych wystąpień superklasy i zbioru zmiennych zadeklarowanych przez podklasę (nazwy nowych zmiennych muszą różnić się od nazw wszystkich zmiennych superklasy). Wystąpienia podklasy mogą korzystać z nowych zdefiniowanych dla niej metod. Jeśli selektor (nazwa) nowej metody jest identyczny z selektorem jednej z metod superklasy, to odpowiednia metoda superklasy jest **wyłączana** (ang. *overriden*), czyli wystąpienia podklasy będą odpowiadać na komunikat z takim selektorem wykonując metodę zdefiniowaną w podklasie.

Gdy adresat będący wystąpieniem pewnej klasy otrzymuje komunikat, musi określić metodę, która powinna zostać wykonana. Proces poszukiwania odpowiedniej metody rozpoczyna się w klasie adresata. Jeśli metoda o selektorze zgodnym z selektorem komunikatu jest zdefiniowana w tej klasie, to przeszukiwane są metody superklasy, następnie metody superklasy superklasy, itd. aż do znalezienia odpowiedniej metody lub stwierdzenia, że taka metoda nie jest zdefiniowana w klasie **Object**. Jeśli odpowiednia metoda nie została znaleziona nigdzie w łańcuchu hierarchii klas, do adresata w zamian zostaje wysłany odpowiedni komunikat sygnalizujący błąd.

Pewna modyfikacja opisanego protokołu znajdowania odpowiedniej metody jest niezbędna w przypadku, gdy metoda pewnej klasy chce skorzystać z metody zdefiniowania w łańcuchu superklas, a omawiana klasa zawiera metodę z identycznym selektorem (problem unikania nieskończonych pętli rekurencyjnych). W takim przypadku rozróżnienie klasy, od której powinno rozpocząć się poszukiwanie odpowiedniej metody, uzyskuje się stosując jedną z dwóch pseudozmiennych, zdefiniowanych dla każdego obiektu. Wysłanie komunikatu z pseudozmienną **self** jako adresatem powoduje przeszukiwanie łańcucha superklas poczynając od klasy adresata, zaś komunikat z adresatem pseudozmienną **super** rozpoczyna poszukiwanie odpowiedniej metody od bezpośredniej superklasy adresata.

Warto wspomnieć, że ograniczenie dziedziczenia do podziału na podklasy nie jest tak mocne, jak wydaje się na pierwszy rzut oka. Jeśli dwie klasy mają wspólną część opisu, a żadna nie jest właściwą podklasą drugiej, można je zawsze powiązać przez wprowadzenie ich wspólnej superklasy definiującej wspólne własności. Taka superklasa nazywa się **superklasą abstrakcyjną** (ang. *abstract superclass*), ponieważ, na ogół, nie ma żadnych wystąpień. Przykładem superklasy abstrakcyjnej jest klasa **Object**, reprezentująca wspólne własności wszystkich obiektów systemu.

Wszystkie składowe systemu Smalltalk-80 są reprezentowane jako obiekty, a każdy obiekt jest wystąpieniem pewnej klasy. Wynika stąd, że sama klasa jest też reprezentowana przez wystąpienie pewnej klasy zwanej **metaklasą** (ang. *metaclass*). Każda klasa jest jedynym wystąpieniem swej metaklasy, tworzonej razem z utworzeniem danej klasy. Metaklasy nie mają nazw i wszystkie są wystąpieniami wspólnej abstrakcyjnej klasy systemowej **Class**. Komunikaty wysyłane do metaklasy są na ogół związane z tworzeniem i zapoczątkowaniem wystąpień klasy oraz inicjalizują zmienne klasowe. Środowisko programowe Smalltalku-80 dostarcza środków do kodowania metod metaklasy wspólnie z metodami odpowiadającej jej klasy.

Klasy systemowe

W systemie Smalltalk-80 zdefiniowano blisko sto klas systemowych z ponad 1000 metodami (dokładna liczba klas i metod zależy od wersji systemu, o czym piszemy dalej). Duża liczba klas i metod wiąże się z tym, że Smalltalk-80 zawiera wiele modułów, zwykle wiązanych z systemem operacyjnym, takich jak: automatyczne zarządzanie pamięcią, system utrzymania plików, obsługa ekranu, edycja tekstów i obrazów, obsługa klawiatury i myszy, kolejkowanie procesów, kompilacja i dekompilacja itd.

Podstawową rolę w systemie spełnia poddrzewo hierarchii klasowej, które opisuje obiekty proste, będące najczęściej składnikami innych obiektów i dające się uporządkować liniowo. Struktura klasy **Magnitude** (wielkość) jest następująca:

Magnitude	wielkość;
Character	znak;
Date	data;
Number	liczba;
Float	liczba rzeczywista;
Fraction	ułamek;
Integer	liczba całkowita;
LargeNegativeInteger	wielka ujemna liczba całkowita;
LargePositiveInteger	wielka dodatnia liczba całkowita;
SmallInteger	mała liczba całkowita (bajt);
Time	czas zegarowy.

System Smalltalk-80 dopuszcza liczby całkowite dowolnej wielkości (klasy **LargeNegativeInteger** i **LargePositiveInteger**) oraz dokładne ułamki, z licznikami i mianownikami o dowolnej wielkości. Wystąpienia klas **Date** i **Time** reprezentują daty i czasy zegarowe.

Zestawienie własności klasy Collection i jej podklas

	Porządek	Rozmiar stały	Duplikaty	Klucze	Klasa elementów
Collection (*)	(x)	(x)	(x)	(x)	(x)
Bag	nie	nie	tak	brak	dowolne
IndexedCollection (*)	tak	(x)	(x)	całkowite	(x)
FixeSizeCollection (*)	tak	tak	(x)	całkowite	(x)
Array	tak	tak	tak	całkowite	dowolne
Bitmap					
ByteArray	tak	tak	tak	całkowite	bajty
CompiledMethod					
FileControlBlock					
Interval	wewnętrzny	tak	nie	całkowite	liczby
String	tak	tak	tak	całkowite	znaki
Symbol					
OrderedCollection	tak	nie	tak	całkowite	dowolne
Process					
SortedCollection	wewnętrzny	nie	tak	całkowite	dowolne
Set	nie	nie	nie	brak	dowolne
Dictionary	nie	nie	nie	dowolne	dowolne
IdentityDictionary					
MethodDictionary					
SystemDictionary					
SymbolSet	nie	nie	nie	brak	dowolne

Trudno jest w tak krótkim tekście opisać własności wszystkich klas systemowych. Dla przykładu i pokazania mocy systemu wybraliśmy klasę systemową **Collection**, której wystąpienia reprezentują agregaty (kolekcje) obiektów. Ogólnie każdy rodzaj kolekcji można określić za pomocą czterech atrybutów:

- porządek – zewnętrzny (klucz), wewnętrzny (zawartość) lub brak uporządkowania;
- rozmiar – stały lub zmienny;
- duplikaty – dopuszczalne lub nie;
- klucze – dostępność do elementów za pomocą kluczy lub brak takiej dostępności.

Pokazana niżej tabela sumuje własności klasy **Collection** i jej podklas. Wcięcia reprezentują hierarchię podklas.

Kolasy oznaczone symbolem (*) są klasami abstrakcyjnymi, symbol (x) oznacza, że klasyfikacja nie odnosi się do danej klasy, a brak opisu własności oznacza, że jest on taki sam, jak dla klasy poprzedzającej. Podklasy, których atrybuty w tabeli pokrywają się, różnią się dodatkowymi cechami wtórnymi. Na przykład, podklasa **IdentityDictionary** musi mieć jako klucze obiekty unikalne.

Składnia wyrażeń

Wyrażenia Smalltalku-80 mogą występować w dwóch kontekstach: w metodach oraz w tekstach pisanych na ekranie. Wyrażenia w metodach są ewaluowane, gdy adresat komunikatu wykonuje odpowiednią metodę ze swej klasy. Użytkownik systemu może również napisać lub wybrać dowolne wyrażenie na ekranie i wydać systemowi zlecenie wykonania wybranego wyrażenia. Wynikiem ewaluacji wyrażenia jest zawsze pewien obiekt.

Wyrażenia mogą składać się z elementów czterech typów:
- stałych – liczb (np. – 33, 1.5E-5), znaków (np. $A, $*), łańcuchów znakowych (np. 'Smalltalk'), symboli (np. #Smalltalk, z których każdy w odróżnieniu od łańcuchów istnieje tylko w jednym egzemplarzu) i tablic stałych (np. #(1 'jeden' #jeden $1));
- identyfikatorów zmiennych dostępnych w danym kontekście; zmiennym można przypisywać wartości stosując tzw. **przedrostki przypisania**, złożone z identyfikatora i znaku < – (w niektórych wersjach także :=), np. **indeks < – cośTam**;
- **wyrażeń selektorowych** opisujących komunikaty skierowane do adresatów;
- **wyrażeń blokowych** opisujących działania odłożone w czasie.

Każde wyrażenie musi zawierać opis adresata i selektor komunikatu. Może także zawierać argumenty, jeśli potrzebuje ich selektor. Na przykład w wyrażeniu:

alfa sin

adresatem jest liczba, do której odnosi się zmienna **alfa**, a selektorem jest **sin**. Adresat decyduje jak odpowiedzieć na komunikat (w tym przypadku jak obliczyć funkcję sinus). W wyrażeniu:

a < – 3 + 4

selektorem jest +, liczba 3 jest adresatem a liczba 4 – argumentem.

Dodatkowo, dzięki przedrostkowi przypisania, odpowiedź adresata na komunikat jest tu przypisywana zmiennej **a**. Smalltalk dopuszcza trzy rodzaje komunikatów:
- **komunikaty unarne**, które nie mają argumentów;
- **komunikaty binarne**, z jednym argumentem, ich selektory mogą się składać z jednego lub dwóch znaków niealfanumerycznych;
- **komunikaty ze słowem kluczowym**, z jednym lub więcej argumentami.

Przykładem wyrażenia z dwoma argumentami jest:

table at: 3 put: 'trzeci'

które w elemencie nr 3 kolekcji, do której odniesieniem jest zmienna **table** umieszcza łańcuch **trzeci**. Nazwą selektora tego komunikatu jest **at:put:**.

Odpowiedź adresata na przesłany mu komunikat może być adresatem lub argumentem innego komunikatu. Na przykład wyrażenie:

alfa tan rounded

zaokrągla wyniki obliczenia tangensa dla obiektu, do którego odnosi się **alfa**, zaś wyrażenie

frame width + border width

oblicza sumę obiektów będących odpowiedziami na komunikaty **frame width** oraz **border width**. Kolejność ewaluacji wyrażeń jest określona następującymi regułami:
- wyrażenia unarne są ewaluowane od lewej do prawej,
- wyrażenia binarne są ewaluowane od lewej do prawej,
- wyrażenia binarne mają pierwszeństwo przed wyrażeniami ze słowami kluczowymi;
- wyrażenia unarne mają pierwszeństwo prze binarnymi;
- wyrażenia w nawiasach okrągłych mają pierwszeństwo przed unarnymi.

Zgodnie z powyższymi regułami np. wyrażenie

3 + 4 * 2

daje odpowiedź 14. Należy więc uważać i stosować nawiasy.

Ostatnią kategorię wyrażeń Smalltalku-80 stanowią bloki. Bloki reprezentują wyrażenia odroczone, które są wykonywane dopiero po otrzymaniu specjalnych komunikatów. Wyrażenia blokowe składają się z ciągu wyrażeń podzielonych kropkami, umieszczonego w nawiasach kwadratowych. Na przykład wyrażenie:

blok < – [indeks < – indeks + 1. table at: indeks put: 0]

przypisuje zmiennej **blok** odniesienie do ciągu wyrażeń w nawiasach kwadratowych. Ciąg ten zostanie wykonany dopiero wtedy, gdy obiekt będący wartością zmiennej **blok** otrzyma komunikat unarny **value**. Bloki mogą mieć argumenty, które deklaruje się na ich początku, poprzedzając ich nazwy dwukropkiem i oddzielając je od treści bloku pionową kreską, np.:

[:nowy | indeks < – indeks + 1. table at: indeks put: nowy]

Takie bloki są wykonywane po otrzymaniu komunikatu **value:**, **value:value:**, itd. z odpowiednią liczbą argumentów, które są podstawiane w miejsce argumentów bloku. Mechanizm blokowy jest stosowany w Smalltalku-80 do implementacji struktur sterujących. Na przykład wyrażenie:

```
(a < )
   ifTrue: [b < – a]
ifFalse: [a < – b]
```

powoduje, że obie zmienne przyjmują wartości min(a, b). Inne struktury sterujące mają podobną składnię.

Środowisko programowe

System Smalltalk-80 jest interakcyjnym graficznym środowiskiem programowym, w którym użytkownik oddziałuje za pomocą rejestrowego ekranu graficznego, klawiatury i urządzenia wskazującego (np. myszy). Reprezentantem urządzenia wskazującego na ekranie jest **kursor**. Ekran może zawierać jedno lub więcej prostokątnych **okien**, których zawartość może być powiązana w sposób wybrany przez użytkownika. Poszczególne okna mogą być nakładane na siebie. Spośród wielu okien jedno (specjalnie podświetlone) jest zwane **oknem aktywnym**. Użytkownik systemu może zmieniać stan aktywności okna, wybierając kursorem jedno z widocznych okien i naciskając odpowiedni guzik urządzenia wskazującego. Inny guzik urządzenia wskazującego służy do przywoływania **wyskakujących menu** (ang. *pop-up menu*) w oknie aktywnym. Repertuar operacji menu zależy od rodzaju okna i **panelu** – prostokątnego fragmentu okna, w którym znajduje się kursor.

Każde okno może składać się z wielu paneli. Różne typy paneli spełniają różne funkcje, związane z typem prezentowanych w nich informacji. Podstawowym typem panelu jest **edytor tekstu**. Panel edytora tekstu pozwala wybierać określone przez użytkownika miejsca w tekście, dopisywać i zmazać fragmenty, zapisywać tekst w pamięci dyskowej itp. Użytkownik może także wybrać fragment tekstu i wydać systemowi zlecenie wykonania go jako wyrażenia. Wynik wykonania

wyrażenia jest pokazany w panelu tekstowym. Innym typem panelu jest **panel listowy**, służący do wybierania informacji przez użytkownika. Najczęściej panel listowy jest powiązany z panelem edytora tekstu w tym samym oknie. Wybranie informacji w panelu listowym wpływa wtedy na odpowiedni wybór tekstu w edytorze. Wśród pozostałych typów paneli systemowych Smalltalko-80 należy wymienić **panel graficzny**, którego otoczka programowa dostarcza dobrych narzędzi do tworzenia grafiki rastrowej. Tworzenie programowych otoczek zastosowań ułatwiają inne typy paneli, a wśród nich **podpowiadacz** (ang. *prompter*), podpowiadając użytkownikowi prawidłowe odpowiedzi i pozwalający je modyfikować oraz menu.

Okna **szperaczy hierarchii klasowej** (ang. *class hierarchy hroswer*) służą do tworzenia nowych klas oraz do modyfikacji klas już istniejących. Takie okno składa się z dwóch paneli listowych, reprezentujących nazwy klas i metod, oraz z panelu tekstowego, który służy do definiowania tekstu nowych metod oraz do modyfikacji metod uprzednio zdefiniowanych. Wszystkie metody są od razu kompilowane przez wbudowany kompilator języka Smalltalk-80, a w bibliotece źródłowej (dostępnej dla szperacza) zachowywane są ich wersje źródłowe. Warto tu wspomnieć, że system Smalltalk-80 niemal w całości może zostać zmodyfikowany przez użytkownika. Jedyne jego fragmenty, których modyfikacja jest trudniejsza i wymaga programowania w języku asemblera, to metody wbudowane, realizujące najbardziej podstawowe funkcje systemu.

Okna szperaczy plikowych dają dostęp do systemu plikowego komputera, na którym zainstalowano Smalltalku-80. Za ich pomocą można modyfikować zawartość dowolnych plików oraz przenosić informacje do i ze Smalltalku-80. Wielką pomoc w uruchamianiu zastosowań stanowi **system uruchomieniowy**, w skład którego wchodzą **okna inspekcyjne** (ang. *inspector*), pokazujące zawartość pamięci poszczególnych obiektów oraz **okna uruchomieniowe** (ang. *debugger*), dające dostęp i możliwość modyfikacji obiektów oraz ich metod w przypadku wystąpienia błędu.

Osobną kategorię środków programowych środowiska Smalltalk-80 stanowią te, które są przeznaczone do aktualizacji systemu. Należą do nich narzędzia wykonujące automatyczny zapis przebiegu sekcji, środki porządkujące źródłową bibliotekę metod oraz metody klasy **pączkowania** (ang. *cloning*), pozwalające wygenerować nową „czystą" kopię systemu, pozbawioną śladów prac programowych.

Dostępność

Smalltalk-80 działa na niemal wszystkich typach komputerów osobistych i stacji roboczych wyposażonych w ekrany graficzne i urządzenia wskazujące (myszy) [4]. Dla komputerów typu IBM PC dostępnych jest kilka wersji tego systemu, stworzonych przez firmę Digitalk Inc. z Los Angeles: Smalltalk/V – dla komputerów klasy XT z ograniczeniem pamięci roboczej do 640 KB; Smalltalk/V 286 – dla komputerów klasy AT, wykorzystujący rozszerzenia pamięci powyżej 640 KB (obie wersje działają w systemie MS DOS); Smalltalk/V PM działający w systemie OS/2. Ta ostatnia wersja umożliwia konstruowanie zastosowań, mogących działać poza środowiskiem Smalltalku (pliki typu .EXE). Środowisko programowe systemu Smalltalk/V [5] różni się pewnymi, niezbyt istotnymi szczegółami od środowiska systemu Smalltalk-80, chociaż oba systemy stosują ten sam język programowania.

Systemy czasu rzeczywistego a języki obiektowe

Wiele współczesnych systemów czasu rzeczywistego oparto na modelu Hoare'a **komunikujących się procesów sekwencyjnych** (KSPS) [3]. Warto zauważyć, że w językach obiektowych jest stosowana terminologia zbliżona do używanej w modelu KSPS. Przede wszystkim, można – po odpowiednim rozwiązaniu problemów implementacyjnych – znaleźć analogie między procesem sekwencyjnym KSPS a obiektem wykonującym właściwie skonstruowaną metodę. Poza tym, można także łatwo zinterpretować w języku KSPS pojęcie wysłania komunikatu do określonego adresata jako komunikację między dwoma procesami. W obu terminologiach różne obiekty (procesy) są niezależne od siebie i nie dzielą wspólnej pamięci. Wszystkie przytoczone argumenty wskazują, że interpretacja modelu KSPS w języku obiektowym powinna być szczególnie łatwa i intuicyjna. System czasu rzeczywistego wymaga współdziałania wielu niezależnych urządzeń sprzętowych. Zwykle takie urządzenia są stosowane w wielu wersjach dostosowanych do sterowanego obiektu. Pojęcie klasy w językach obiektowych ułatwia specyfikację generycznych typów urządzeń sprzętowych i ich uszczegółowianie w konkretnym zastosowaniu.

Nie do pogardzenia jest również przydatność niektórych języków obiektowych (w tym także języka Smalltalk-80) do implementacji procesów współbieżnych chociaż maszyna wirtualna Smalltalku-80 jest z założenia maszyną jednoprocesową. Cecha ta pozwala w miarę łatwo konstruować mechanizmy symulacyjne systemów czasu rzeczywistego. Symulacja zaś ułatwia poprawienie wielu błędów już podczas projektowania systemu czasu rzeczywistego. Ze względu na rozmiary związane ze środowiskiem programowym, system Smalltalk-80 nie nadaje się do docelowej implementacji systemów czasu rzeczywistego. Może jednak służyć jako wygodne narzędzie do projektowania takich systemów.

Środowisko do projektowania systemów czasu rzeczywistego

Projektowanie systemów czasu rzeczywistego wymaga przyjęcia określonych założeń. Jako założenie podstawowe przyjmujemy:

Założenie 1: Procesy sterujące pracą poszczególnych urządzeń systemu czasu rzeczywistego współpracują zgodnie z regułami modelu KSPS.

Większość znanych autorom niewielkich systemów czasu rzeczywistego jest projektowana i kodowana „na piechotę", bez zastosowania jakichkolwiek narzędzi projektowych. Taki sposób projektowania powoduje często nieporozumienia pomiędzy technologami, odpowiedzialnymi za przebieg sterowanych procesów technologicznych, których wiedza informatyczna jest na ogół znikoma, a projektantami systemu – informatykami, którzy zwykle nie do końca rozumieją założenia technologiczne. Nieporozumienia te wpływają na przebieg procesu projektowania. Przyjmujemy więc:

Założenie 2: Projekt systemu powinien mieć dwie warstwy: słowną – dostępną do modyfikacji przez technologów oraz programową – projektowaną przez współpracujących z nimi informatyków. Schematy logiczne procesów powinny być warażane najprostszymi technikami graficznymi (np. za pomocą diagramów blokowych).

Kolejne założenie wiąże się z obserwacją, że procesy obsługi urządzeń kontaktujących się ze światem zewnętrznym są w różnych zastosowaniach zbliżone (np. procesy obsługi układów wejść binarnych są wszystkie bardzo podobne, a różnią się tylko takimi szczegółami, jak liczba wejść w układzie sprzętowym). Projektowanie ułatwia więc:

Założenie 3: Programy i schematy logiczne procesów powinny poddawać się łatwemu przenoszeniu między różnymi projektami. Można to osiągnąć np. przez stworzenie biblioteki typowych procesów.

Środowisko projektowe powinno ułatwiać szybkie budowanie i testowanie prototypów projektowanych systemów. W związku z tym przyjęto:

Założenie 4: środowisko programowe powinno zawierać odpowiednie środki do symulacji projektowanych systemów z wizualizacją graficzną ich pracy.

Pozostałe założenia dotyczące środowiska projektowanego są typowe i przewidują m.in. jego prostotę, autodokumentację projektów, proste metody przechowywania projektów w pamięci dyskowej itp. W związku z decyzją o zastosowaniu Smalltalku-80 jako podstawy środowiska programowego przyjmujemy również założenie, że językiem źródłowym, w którym są wyrażane projekty procesów, jest sam język Smalltalk-80. Założenie to można jednak pominąć uzupełniając środowisko projektowe o translator jakiegokolwiek prostego języka programowania o mocy wystarczającej do programowania procesów sekwencyjnych.

Graficzne środowisko projektowe

Na środowisko projektowe składa się system sprzężonych między sobą okien na trzech poziomach:
(a) poziom definiowania modeli systemów czasu rzeczywistego,
(b) poziom definiowania urządzeń,
(c) poziom definiowania procesów dla urządzeń.

Poziom (a) służy do identyfikacji modeli (nazwa, autorzy, komenta-

rze, data ostatnich zmian) oraz pozwala wybierać kartoteki dyskowe, w których będą przechowywane modele. Na tym poziomie odbywa się również ładowanie wybranego lub świeżo zdefiniowanego modelu oraz jego zapamiętywanie na dysku. Zewnętrzna reprezentacja modeli jest czysto tekstowa (pliki ASCII), co pozwala łatwo przenosić modele do innych systemów komputerowych.

Okna poziomu (b) służą do definiowania urządzeń programowych stanowiących abstrakcję urządzeń sprzętowych stosowanych w systemach czasu rzeczywistego. Projektant dysponuje tu prototypami typowych klas urządzeń (procesor, konsola, zegar, wejścia–wyjścia binarne, wejścia–wyjścia analogowe itp.) wraz ze wcześniej zdefiniowanymi typowymi procesami obsługi. Może także definiować nowe klasy urządzeń, w tym także precyzować charakterystyczne cechy urządzeń stosowanych w projektowanym systemie czasu rzeczywistego. Na przykład, jeśli przewidziano zastosowanie konsoli zgodnej ze standardem VT-100, projektant może stworzyć podklasę klasy urządzeń **Konsola** i zmodyfikować odpowiednie procesy obsługi, wprowadzając odpowiednie reakcje systemu na ciągi znaków sterujących standardu VT-100. Procesy klasy **Procesor**, w całości definiowane przez projektanta, obsługują główny procesor (procesory) projektowanego systemu czasu rzeczywistego, koordynując współdziałanie procesów obsługi innych urządzeń programowych. Na poziomie (b) można zdefiniować dowolną liczbę urządzeń programowych danej klasy nadając im odpowiednie nazwy.

Procesy obsługi urządzeń programowych projektuje się w oknach na poziomie (c). Projektowany proces jest reprezentowany graficznie w postaci diagramu blokowego o strukturze dopuszczającej wyłącznie projektowanie metodą *top-down*. Projektowanie procesu polega na zdefiniowaniu i powiązaniu elementów graficznych reprezentujących poszczególne instrukcje sekwencyjne, bloki, pętle i diagramy warunkowe. Obraz graficzny każdego elementu ma dwie warstwy: opisową oraz programową. W warstwie programowej można tworzyć, uruchamiać i niszczyć inne procesy obsługi urządzeń programowych, a także przekazywać im komunikaty (według protokołu KSPS). Projektant dysponuje środkami umożliwiającymi edycję graficzną projektu. Może m.in. przenosić fragmenty lub całe reprezentacje procesów między różnymi modelami lub urządzeniami. Odpowiednie menu ułatwiają przygotowanie dokumentacji projektu z zastosowaniem drukarki lub plotera.

Symulacja systemów czasu rzeczywistego

W skład systemu projektowania wchodzą także elementy ułatwiające symulację i testowanie projektów. Dla każdej klasy urządzeń programowych w podsystemie symulacyjnym zdefiniowano odpowiednie reprezentacje graficzne urządzeń, które ułatwiają wizualizację symulowanych procesów obsługi. Po uruchomieniu podsystemu symulacyjnego projektant może oddziaływać na pracę poszczególnych urządzeń programowych w sposób przewidziany przez ich procesy obsługi. Może także skorzystać z przygotowanego wcześniej skryptu definiującego oddziaływanie „świata zewnętrznego" z symulowanym systemem czasu rzeczywistego. Skrypt definiuje się jako proces obsługi dodatkowego urządzenia programowego klasy **świat** wymieniającego komunikaty z innymi urządzeniami programowymi.

Uwagi implementacyjne

System projektowania został zaimplementowany w Smalltalku/V, a więc wersji przeznaczonej dla mikrokomputerów klasy IBM PC. Trudności implementacyjne wiązały się głównie z dwoma problemami: wielkością pamięci operacyjnej oraz przełączaniem zadań (ang. *switching*).

Zastosowana wersja Smalltalku nie wykorzystuje rozszerzeń pamięci. Pod względem wykorzystania pamięci Smalltalk-80 jest bardzo rozrzutny. Szczególnie dużo miejsca pamięci operacyjnej zajmują graficzne obrazy okien oraz inne obiekty związane ze środowiskiem graficznym. W związku z tym, projektowanie wymagało bardzo oszczędnego przydziału pamięci dla innych obiektów. Przyjęte rozwiązania spowodowały spowolnienie odpowiedzi systemu projektowania. Jest to szczególnie widoczne podczas przesuwania (ang. *scrolling*) zawartości okien graficznych. Ciągle jednak reakcja systemu mieści się w granicach tolerancji operatora, jeśli zegar taktujący używanego mikrokomputera jest wystarczająco szybki (co najmniej 8 MHz).

W odróżnieniu od „standardowego" Smalltalku-80, w Smalltalk-/V brak jest niezbędnego podczas symulacji mechanizmu przełączania zadań. Brak ten uzupełniono wprowadzając dodatkowe klasy systemowe. Podczas implementacji okazało się jednak, że wprowadzone mechanizmy w pewnych przypadkach zawodzą. Przyczyną jest prawdopodobnie złe współdziałanie niskopoziomowych procedur Smalltalku/V z układem przerwań komputera IBM PC. Problem ten ciągle nie został rozwiązany do końca.

LITERATURA

[1] Goldberg A., Robson D.: Smalltalk-80: The Language and Its Implementation. Addison-Wesley, Reading, Massachusetts, 1983

[2] Goldberg A.: Smalltalk-80: The Interactive Programming Environment. Addison-Wesley, Reading, Massachusetts, 1984

[3] Hoare C.A.R.: Communicating Sequential Processes. CACM 21 (1978), p. 666

[4] Kaehler T., Patterson D.: A Taste of Smalltalk. W.W. Norton & Company, New York, 1986

[5] Smalltalk/V, Tutorial and programming Handbook. Digitalk Inc., 1986.

KARTY PRZETWORNIKÓW
ANALOGOWO-CYFROWYCH
do komputerów IBM XT/AT/386
Podzespoły ANALOG DEVICES
MAXIM BURR BROWN

- 12, 14, 22 bity
- izolacja galwaniczna
- pamięć buforowa
- praca „w tle"

próbkowanie do 1 MHz!!!

KARTY PROCESORÓW SYGNAŁOWYCH
TMS320C25 MOTOROLA 56001 AT&T DSP32C

Bogate oprogramowanie
Kompleksowa obsługa zamówień
Serwis ★ Gwarancja ★ Pomoc w instalacji

PRACOWNIA USŁUG INFORMATYCZNYCH
I KOMPUTEROWYCH 'CONVERT'
50-412 Wrocław, ul. Mazowiecka 17,
telefon 300-11 w. 210, telefaks 44-85-66

O/4/91

SIMPLE ® Sp. z o.o.

to organizacja + informatyka
to największy w Polsce
Software House

Oferujemy wszystkie dostępne na rynkach zachodnich pakiety oprogramowania licencjonowanego po bardzo niskich cenach. I tak np.:

Microsoft
- Quick C wersja 2.0 – 1,5 mln zł
- Windows wersja 3.0 – 2,5 mln zł

Borland International
- Turbo PASCAL wersja 6.0 Professional – 3,9 mln zł
- Turbo C wersja 3.0 Professional – 3,9 mln zł

Peter Norton Computing
- Norton Commander wersja 3.0 – 2,1 mln zł
- Norton Utilities wersja 5.0 – 2,5 mln zł

Nantucket
- Clipper wersja 5.0 –7,5 mln zł

Pełny katalog oprogramowania (aktualizowany co miesiąc) oferujemy w cenie 50 tys. zł lub udostępniamy w siedzibie firmy. Realizujemy każde zamówienie w terminie do trzech tygodni.

Oferujemy poza tym własne oprogramowanie użytkowe przedsiębiorstw, przeznaczone do pracy w sieci firmy Novell, opracowane na podstawie oryginalnej bazy danych SBase, nowoczesne narzędzia – język C, pakiet dialogu Vermont Views. Zapewniamy szkolenie i wersje źródłowe oprogramowania.

Pamiętaj: Simple to dobry business!

SIMPLE
Sp. z o.o.
04-541 Warszawa
ul. Karpacka 14 c

Zakład
Informatyki
ul. Karpacka 12
tel.: 15-49-83
faks: 635-91-66

SO/84/91

EWA GRABSKA
AGNIESZKA POLARSKA
Katedra Informatyki
Uniwersytet Jagielloński
Kraków

Rola komputera w ocenie estetycznej kształtów

Istnieją pewne dziedziny, np. sztuka, wokół których toczy się dyskusja co do celowości stosowania w nich komputerów. Dla niektórych komputer jest nowym, obiecującym narzędziem pracy, inni uważają, że pozbawia on artystę swobody twórczej.

Z działalnością twórczą jest związana ocena jej efektów. Podstawowym warunkiem umożliwiającym automatyczną ocenę jest istnienie miary estetycznej obliczalnej algorytmicznie. Wzór na obliczanie takiej miary przedstawił w latach trzydziestych amerykański matematyk Birkhoff w swojej systematycznej teorii piękna [1].

W artykule zaprezentowano zarys teorii Birkhoffa oraz opis programu służącego do obliczania miary estetycznej [4, 5]. Obiektami poddawanymi ocenie estetycznej są wielokąty, ze względu na to, że mają prostą reprezentację komputerową, a jednocześnie stanowią klasę obiektów, w której nie występuje z góry narzucony porządek. Na zakończenie pokazano, w jaki sposób można rozszerzyć miarę estetyczną Birkhoffa na pewną klasę oranmetnów i przedstawiono wnioski dotyczące zastosowanej metody wartościowania estetycznego.

Wielokąt jako przedmiot doznań estetycznych

Od niepamiętnych czasów wielokąty są używane do konstruowania geometrycznych wzorów dekoracyjnych, a także rozpatrywane w aspekcie abstrakcyjnych form geometrycznych. Konsekwencją stosowania wielokątów jako elementów dekoracyjnych były próby ich oceny estetycznej, np. wielokątom foremnym z reguły przypisywano najwyższe oceny estetyczne.

Do czynników, które wpływają dodatnio na wartość estetyczną wielokątów należy z całą pewnością symetria. Wielokąt wydaje się szczególnie atrakcyjny, gdy ma pionową oś symetrii.

Istotnym czynnikiem jest także równowaga wielokąta. Chodzi o to, aby środek ciężkości był umieszczony w odpowiednim miejscu. Rozmieszczenie boków wielokąta na prostych prostopadłych lub równoległych również zwiększa walory estetyczne.

Wielokąty wypukłe mają z reguły wyższą ocenę estetyczną od wklęsłych. Ocena wielokąta wklęsłego będzie tym wyższa, im mniejsza różnorodność nisz będzie w nim występować (*niszę definiuje się jako obszar leżący na zewnątrz wielokąta wklęsłego, ale wewnątrz obwiedni wypukłej*). Wśród wielokątów wklęsłych są również preferowane takie, w których dla każdego boku wklęsłego (tzn. tworzącego wklęsłość) istnieje inny bok leżący z nim na tej samej prostej.

Ujemnie na ocenę estetyczną wielokąta może wpłynąć długość boku lub wielkość kąta. Jeśli – na przykład – trójkąt ma zbyt krótki bok lub kąt bliski 180°, to staje się podobny do odcinka. Występuje tutaj problem niejednoznaczności w określeniu figury. Obserwacja wielokątów o liczbie boków większej niż sześć dostarcza dalsze czynniki wpływające na miarę estetyczną. Wartość estetyczna wielokąta zmniejsza się wraz ze wzrostem liczby linii wyznaczonych przez jego boki. Zauważalny jest związek wielokątów z równomierną poziomo-pionową siatką. Tego rodzaju zależność ma pozytywny wpływ na doznania estetyczne. Związek ten może być bezpośredni, jak na przykład w greckim krzyżu i swastyce (rys. 1a, 1b) lub pośredni, gdy wielokąt jest związany z siatką romboidalną, tzn. z siatką utworzoną przez linie nachylone pod tym samym kątem do pionu (rys. 2). Zakłada się, że w ocenie estetycznej wielokątów nie będą brane pod uwagę funkcje semantyczne związane z figurami. Funkcje te niewątpliwie mogłyby wpłynąć na ocenę (np. w przypadku krzyża lub swastyki).

Wielokąty stanowią obiekty, do których Birkhoff w pierwszej kolejności zastosował swoją teorię piękna. Określona przez niego miara estetyczna została wyrażona wzorem:

$$M = \frac{porządek}{złożoność} = \frac{0}{C} = \frac{V + E + R + HV - F}{C}$$

w którym porządek 0 zależy od pięciu składowych:
V – symetria względem pionowej osi,
E – równowaga,
R – symetria obrotowa,
HV – relacja do siatki poziomo-pionowej,
F – niedoskonałość formy.
Wszystkie elementy miary estetycznej M mogą być obliczone algorytmicznie. Poniżej podamy wartości, które zostały przypisane poszczególnym składowym miary estetycznej.

C – złożoność

Złożoność C wielokąta jest zdefiniowana jako liczba prostych, które zawierają wszystkie boki wielokąta. Przykładowo, dla czworokąta złożoność wynosi 4, a dla greckiego krzyża (rys. 1a), którego liczba boków jest równa 12, wynosi 8. Wielokąt z rys. 3, również o 12 bokach ma złożoność równą 10. Uzasadnienie takiej definicji złożoności jest następujące: dla wielokątów wypukłych oraz dla wielokątów wklęsłych bez boków, dla których istnieją inne boki leżące z nimi na tych samych prostych, złożoność jest równa liczbie boków. Wysiłek oka śledzącego kontur wielokąta jest proporcjonalny do tej liczby. Jeśli natomiast w wielokącie na jednej prostej z pewnym bokiem znajduje się inny bok, to oko obejmuje te boki w jednym ruchu. Tak, na przykład, rozważany krzyż można postrzegać jako figurę zbudowaną z dwóch prostokątów.

V – symetria względem pionowej osi

Przyjmuje się, że V równa się 1, gdy wielokąt ma pionową oś symetrii, a równa się 0 w przeciwnym razie.

E – równowaga

Równowagę dla wielokąta określa się na podstawie pionowej osi symetrii lub położenia środka ciężkości. Składnikowi E nadaje się następujące wartości liczbowe:
$E = 1$, gdy $V = 1$, tzn. pionowa oś symetrii wielokąta stanowi punkt spoczynku dla oka i dlatego wielokąty mające taką oś są w równowadze;
$E = 1$ jeśli środek ciężkości leży ponad punktem D, położonym na poziomym odcinku AB, będącym podstawą wielokąta, przy założeniu, że długości AD i BD są większe niż 1/6 całkowitej poziomej szerokości wielokąta;
$E = 0$, gdy $V = 0$ i gdy dla środka ciężkości leżącego powyżej punktu D podstawy AB nie jest spełniony warunek, że długości AD i BD są większe niż 1/6 poziomej szerokości wielokąta;
$E = -1$ w pozostałych przypadkach.
Na przykład, dla wielokąta (z rys. 4) $E = 1$, gdyż ma on pionową oś symetrii; dla wielokąta z rys. 3 jest natomiast $E = 0$.

R – symetria obrotowa

Przed omówieniem symetrii obrotowej trzeba wprowadzić pojęcia elementu oraz elementu tego samego typu.

Przez *elementy wielokąta* rozumie się wierzchołki, boki, nisze oraz proste, w których zawierają się boki wielokąta. Dodatkowo, jako elementy wielokąta są traktowane również proste zawierające boki, gdyż wyznaczają one kierunki, tzn. zbiory prostych o tym samym współczynniku kierunkowym.

Elementami tego samego typu co dany element wielokąta są wszystkie obrazy danego elementu otrzymane w wyniku przekształceń należących do grupy symetrii tego wielokąta. Mówi się zatem o wierzchołkach, bokach, niszach i kierunkach tego samego typu.

Najprostszym rodzajem symetrii obrotowej jest symetria środkowa z kątem obrotu 180°. Symetria ta jest od razu zauważalna. Przyjmuje się, że R jest równe 1, gdy występuje symetria środkowa. Wielokąty regularne mają symetrię obrotową o najmniejszym kącie obrotu równym $360°/q$ i rozpadają się na q figur tego samego typu. Zakłada się, że R zmienia się proporcjonalnie do q.

Dla rozpatrywanego powyżej przypadku symetrii środkowej wartość q równa się 2, a więc R definiuje się jako $q/2$. Wrażenie symetrii obrotowej jest kompletne dla q równego lub większego od 6. W takich sytuacjach figury są podobne do okręgów opisanych na nich. Dlatego też R przyjmuje wartość $q/2$, dla q nie większych od 6, oraz wartość 3 dla większych wartości q.

Podsumowując:
$R = \min\{q/2, 3\}$, jeśli dany wielokąt ma symetrię obrotową przy założeniach, że:
- jeśli jest wypukły, to ma też pionową oś symetrii,
- jeśli jest wklęsły, to obwiednia, czyli minimalny wielokąt wypukły zawierający go, ma symetrię pionową, a żaden z wierzchołków tego wielokąta wypukłego nie spoczywa na niszy danego wielokąta (rys. 5 i rys. 1b);

$R = 1$ w każdym innym przypadku gdy q jest parzyste, czyli gdy występuje symetria środkowa. Wielokąty z rys. 6a i 6b mają $R = 1$, gdyż obydwa są środkowo symetryczne, ale w pierwszym z nich (rys. 6a) wierzchołki jego obwiedni spoczywają na niszach, drugi natomiast, który jest wypukły (rys. 6b) nie ma pionowej osi symetrii;
$R = 0$ w pozostałych przypadkach (rys. 7a i 7b).

HV – relacja do siatki poziomo-pionowej

Zgodność wielokąta z siatką może być całkowita lub żadna, a między tymi skrajnymi przypadkami występuje wiele pośrednich. Na ogół twierdzi się, że jeśli dla pewnego wielokąta występuje odchylenie od zgodności z siatką, to przyjemność z jego oglądania maleje.

Najkorzystniejsza sytuacja występuje wówczas, gdy wszystkie boki wielokąta leżą na liniach równomiernej poziomo-pionowej siatki w taki sposób, że wszystkie te linie wypełniają całkowicie minimalny prostokątny fragment siatki zawierający wielokąt. Wtedy przyjmuje się, że HV ma wartość 2. Przykładami wielokątów o takich właściwościach jest wspomniany już krzyż grecki oraz swastyka (rys. 1a, 1b). Wybór 2 jako odpowiadającego wskaźnika dla HV jest spowodowany tym, że istnieją dwa rodzaje niezależnych ruchów translacyjnych przesuwających siatkę do pozycji, które mogą być identyfikowane z wyjściową. Tymi ruchami są translacja w prawo lub w lewo oraz translacja w górę lub w dół. Każda inna translacja jest ich kombinacją.

Jeśli wszystkie boki wielokąta leżą na liniach równomiernej siatki utworzonej przez dwa zbiory linii równoległych, w równym stopniu nachylonych do pionu oraz wszystkie te linie wypełniają całkowicie minimalny romboidalny fragment siatki zawierający wielokąt, to $HV = 1$. Wielokąt z rys. 2 jest właśnie tego rodzaju. Dodatkowo przyjmuje się, że $HV = 1$, jeśli dla wielokąta oraz minimalnego prostokątnego fragmentu siatki zawierającego ten wielokąt są dopuszczalne następujące wyjątki od przypadku, gdy HV = 2:
1) jeden blok wielokąta i inne boki tego samego typu mogą leżeć na przekątnych oczek siatki prostokątnej (rys. 3);
2) na jednej poziomej i jednej pionowej linii siatki, które wypełniają minimalny prostokątny fragmnet siatki zawierającej wielokąt, nie znajdują się boki wielokąta.
Dla wielokąta z rys. 8 $HV = 1$, bo w minimalnym fragmencie siatki zawierającym ten wielokąt istnieje linia siatki, która nie przechodzi przez żaden jego bok. Analogiczne wyjątki dopuszcza się, gdy wielokąt zawiera się w siatce romboidalnej i wówczas również $HV = 1$. Przyjmuje się, że $HV = 0$ dla wszystkich pozostałych przypadków.

F – niedoskonałość formy

F zawiera w sobie wszystkie czynniki ujemnie wpływające na wartość estetyczną wielokąta. Przyjmuje się, że $F = 0$, co jest równoważne z doskonałością formy, gdy są spełnione następujące warunki:
- minimalna odległość każdego wierzchołka od innego wierzchołka lub odległość między równoległymi bokami stanowi co najmniej 1/10 maksymalnej odległości między wierzchołkami wielokąta,
- kąt między każdymi dwoma nierównoległymi bokami jest nie mniejszy niż 20°,
- mogą występować co najwyżej dwa typy kierunków, przy założeniu, że poziomy i pionowy kierunek, jeśli pojawią się oba, są liczone jako jeden,
- wielokąt jest symetryczny, tzn. V i R nie są jednocześnie równe 1.

Dla wielokątów wklęsłych wprowadza się dodatkowo dwa warunki:
- dla każdego boku wklęsłego istnieje inny bok leżący z nim na tej samej prostej,
- występuje co najwyżej jeden typ nisz.

Jeśli jeden i tylko jeden warunek nie jest spełniony, to przyjmuje się, że $F = 1$. W pozostałych przypadkach wartość F wynosi 2. Na przykład krzyż grecki z rys. 1a odznacza się doskonałością formy (F = 0), a wielokąt z rys. 3 ma $F = 1$, ze względu na to, że nie jest spełniony warunek 5.

AES – komputerowy system oceny estetycznej

Teoria Birkhoffa stała się podstawą do opracowania komputerowego systemu oceny estetycznej wielokątów [2]. System ten zwany AES, składa się z dwóch modułów: edytora graficznego wielokątów GPE oraz modułu EM obliczającego miarę estetyczną wielokątów na podstawie wzoru Birkhoffa.

Podstawowymi operacjami edytora GPE są:
- rysowanie wielokąta na ekranie monitora wraz z zapamiętaniem jego reprezentacji wewnętrznej,
- modyfikowanie wielokątów (aktualnie definiowanego oraz znajdujących się w bazie graficznej),
- usuwanie wielokąta z ekranu wraz z jego reprezentacją wewnętrzną.

Moduł EM zawiera algorytmy wartościujące składowe miary estetycznej Birkhoffa zdefiniowanej dla wielokątów. Poniżej przedstawiono zarys metod zastosowanych w algorytmach modułu EM. Szczegółowe opisy algorytmów można znaleźć w [4, 5].

Obliczanie złożoności C

Algorytm obliczający złożoność C korzysta z funkcji podającej liczbę prostych, w których zawierają się boki wielokąta. Dla wielokątów wypukłych liczba boków wyznacza wartość tej funkcji. W przypadku wielokątów wklęsłych dla kolejnych boków sprawdza się, czy istnieje inny bok leżący na tej samej prostej.

Sprawdzanie istnienia symetrii pionowej V

Algorytm badający symetrię V wykorzystuje funkcję pomocniczą sprawdzającą, czy wielokąt jest symetryczny względem osi symetrii $Ax + By + C = 0$. Kolejno są wybierane wierzchołki leżące na prawo i na lewo od domniemanej osi symetrii i jest sprawdzane ich położenie względem tej osi. Istnienie symetrii pionowej sprawdza funkcja główna algorytmu, która wyznacza jedyną możliwą pionową prostą znajdującą się w połowie odległości między maksymalną i minimalną współrzędną x. Następnie wspomniana już funkcja pomocnicza bada, czy wytypowana prosta jest faktycznie osią symetrii.

Sprawdzanie istnienia równowagi E

Funkcja badająca równowagę E sprawdza wartość czynnika V. Jeśli $V = 1$, to $E = 1$. W przeciwnym razie wykorzystuje się obliczone wcześniej współrzędne środka ciężkości i w zależności od tego, jak on jest położony w stosunku do punktów podparcia wielokąta, funkcja przyjmuje wartość 1, 0 lub − 1.

Badanie istnienia symetrii obrotowej R

Algorytm wyznacza możliwe kąty obrotu dla wielokąta, na podstawie liczby punktów równoodległych od środka ciężkości oraz podzielników tej liczby. Następnie znajduje współrzędne wierzchołków po obrocie względem środka ciężkości i porównuje je ze współrzędnymi przed obrotem. Jeżeli dla żadnego kąta współrzędne przed i po obrocie nie pokrywają się, to $R = 0$. W przeciwnym razie w zależności od tego, czy wielokąt jest wypukły czy wklęsły, sprawdza się istnienie osi symetrii dla niego lub jego obwiedni i przyporządkowuje się R odpowiednią wartość.

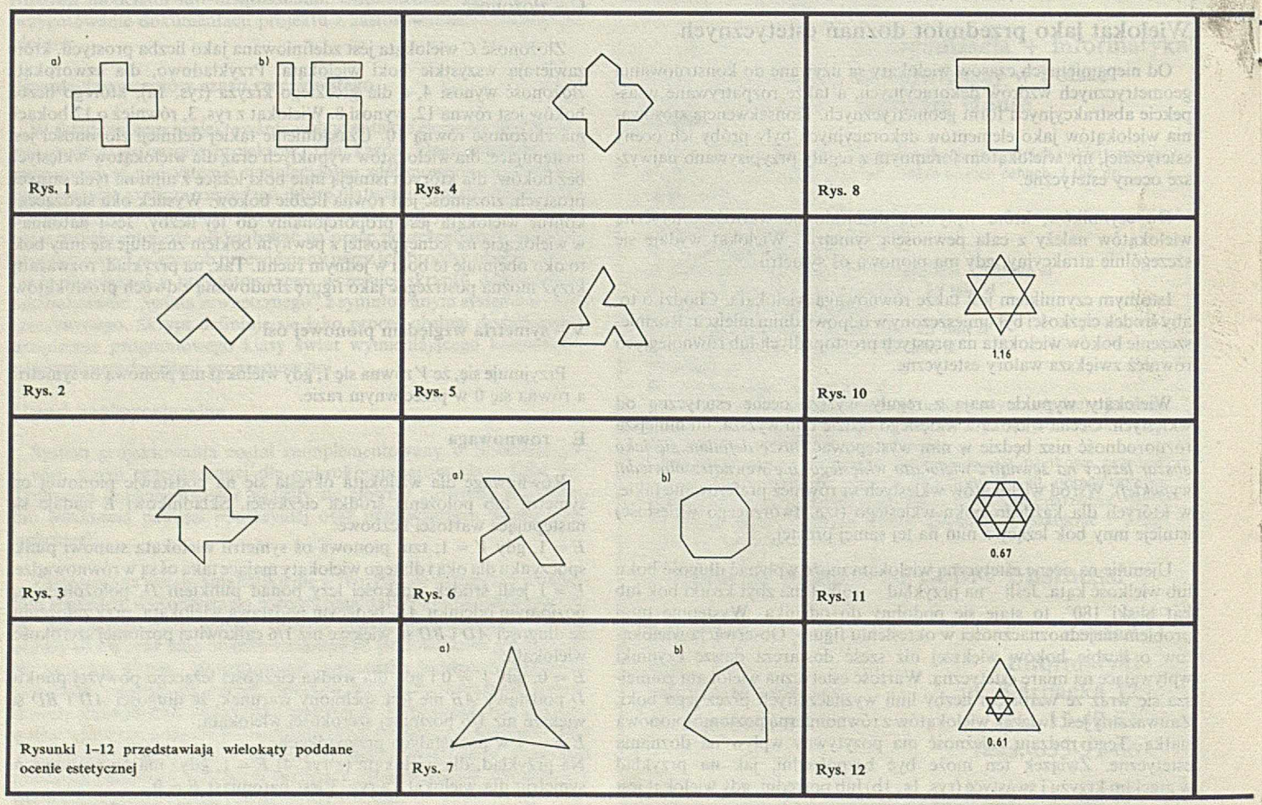
Sprawdzanie relacji do siatki HV

W algorytmie tworzy się trzy listy: dwie zawierające proste o kierunku pionowym i poziomym oraz jedną dla pozostałych prostych (wyznaczanych przez boki wielokąta). Jeśli dla danego wielokąta dwie pierwsze listy zawierają co najmniej dwie proste, to określa się rozmiary wszystkich możliwych oczek siatki poziomo-pionowej i sprawdza się warunki pokrycia siatki przez wielokąt. W przeciwnym razie oraz gdy wartość $HV = 0$, dla siatki prostokątnej, rozważa się siatkę romboidalną.

Badania niedoskonałości formy

Wartość funkcji badającej niedoskonałość formy zależy od wartości jej sześciu funkcji wewnętrznych, które sprawdzają warunki dotyczące formy wielokąta. Wykorzystuje się do tego wcześniej utworzone tablice elementów tego samego typu. W ten sposób unika się nadmiarowości obliczeń. Za pomocą systemu AES obliczono miary estetyczne wielu wielokątów.

Rozważmy figurę znajdującą się na rys. 1b. Jest to swastyka, a jej miara estetyczna wynosi 1/3. Prześledźmy przebieg obliczeń. Złożoność $C = 12$. Swastyka nie ma pionowej osi symetrii, a zatem $V = 0$. Składowa $E = 1$, gdyż figura znajduje się w równowadze optycznej. Najmniejszy kąt obrotu wynosi 90°, zatem $q = 4$. Obwiednia wielokąta jest pionowo symetryczna i żaden z jej wierzchołków nie spoczywa na



Rys. 1

Rys. 2

Rys. 3

Rys. 4

Rys. 5

Rys. 6

Rys. 7

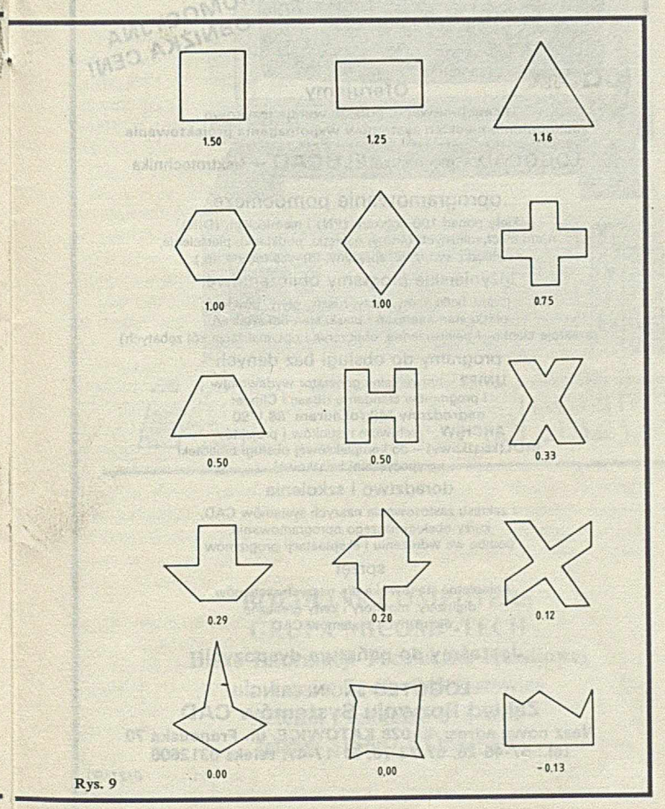
Rys. 8

Rys. 10

Rys. 11

Rys. 12

Rysunki 1–12 przedstawiają wielokąty poddane ocenie estetycznej



Rys. 9

niszy. Stąd $R = \min\{q/2,3\} = 2$. Dla swastyki występuje całkowita zgodność z siatką linii poziomo pionowych, czyli $HV = 2$; $F = 1$, bo dla jednego typu boków nie istnieją inne boki leżące z nimi na tych samych prostych, czyli jeden z warunków nie jest spełniony. Wstawiając obliczone wartości do wzoru na miarę estetyczną otrzymuje się:

$$M = \frac{V + E + R + HV - F}{C} = \frac{0 + 1 + 2 + 2 - 1}{12} = \frac{1}{3}$$

Lista wybranych wielokątów wraz z wartościami ich miar estetycznych obliczonych za pomocą AES znajduje się na rys. 9.

Miara estetyczna ornamentów

Miarę estetyczną Birkhoffa można rozszerzyć na bardziej złożone obiekty estetyczne. Rozważanymi obiektami będą ornamenty, znane już w zamierzchłej przeszłości. Od dawna bowiem był wykorzystywany fakt, że nawet prymitywny rysunek wielokrotnie powtarzany w prawidłowy sposób daje całość przyjemną dla oka. Twórcy niektórych ornamentów egipskich umieli pomysłowymi symbolami podkreślać prawidłowość pewnych powtórzeń i to powtórzeń niebanalnych [3].

Figurę narysowaną, namalowaną, wyrytą lub w inny sposób wykonaną na powierzchni nazywa się *ornamentem*, jeśli istnieje co najmniej jedna nietrywialna transformacja będąca obrotem, translacją, ruchem śrubowym lub odbiciem zwierciadlanym, która przemieszcza figurę do jej początkowego położenia.

Tutaj ograniczymy się do ornamentów prostoliniowych, to znaczy zbudowanych z odcinków linii prostych. Przykładowy ornament prostoliniowy przedstawiono na rys. 10. Wzór na miarę estetyczną M prostoliniowego ornamentu jest następujący:

$$M = \frac{porządek}{złożoność} = \frac{V + E + R + HV + S - F}{C}$$

gdzie V, E, R, HV, F i C są naturalnymi modyfikacjami składowych zdefiniowanych dla wielokątów, a jedynie podobieństwo S wyznacza nową składową.

Przed omówieniem podobieństwa S trzeba wprowadzić pojęcie wielokąta ornamentu. *Wielokątem ornamentu* nazywa się:
- wielokąt elementarny ograniczony przez boki ornamentu, który nie zawiera żadnej części ornamentu wewnątrz niego;
- najbardziej zewnętrzny wielokąt dostępny z zewnętrznej części płaszczyzny;
- wielokąt zbudowany z boków ornamentu, których wierzchołki nie są końcami innych boków ornamentu.

Wielokąty ornamentu mają tę właściwość, że są łatwo zauważalne. Przykładowo, figura z rys. 10 zawiera 10 wielokątów ornamentu. Centralny, regularny sześciokąt oraz sześć małych trójkątów równobocznych są wielokątami elementarnymi pierwszego typu, najbardziej zewnętrzna gwiazda sześcioramienna jest typu drugiego, a dwa duże trójkąty równoboczne są typu trzeciego.

Nowa składowa porządku, będąca podobieństwem S, jest z definicji równa zero, jeśli dla żadnego wielokąta nie istnieje wielokąt w rozważanym ornamencie do niego podobny. Jeśli istnieje relacja podobieństwa tylko dla jednego wielokąta ornamentu, to podobieństwo S przyjmuje wartość 1. W przypadku większej liczby relacji podobieństwa zdefiniowanych dla różnych niepodobnych wielokątów ornamentu $S = 2$. Dla tak zdefiniowanej składowej podobieństwa S można znaleźć algorytm obliczający jej wartości. Przykładowo, ornament przedstawiony na rys. 10 ma miarę estetyczną równą 1,16. Miary estetyczne ornamentów z rys. 11 i 12, zawierających ten ornament, wynoszą odpowiednio 0,67 i 0,61.

* * *

W artykule autorki przedstawiły swój punkt widzenia na rolę, jaką może odegrać komputer w ocenie estetycznej. Istotną zaletą zastosowania komputera – poza oczywistym ułatwieniem wykonywania złożonych obliczeń potrzebnych do wykrywania właściwości wielokątów – jest możliwość gromadzenia w jego pamięci uzyskanych już wyników. Utworzenie takiej bazy danych dla danej klasy obiektów (np. wielokątów) umożliwia szybkie wyszukiwanie wielokątów o żądanej wartości miary estetycznej. Wielokąty takie mogą być wyświetlane na ekranie monitora w celu np. porównania indywidualnych doznań estetycznych z przyjętą metodą wyznaczania miary estetycznej. Programy wyznaczające wartość estetyczną pewnych klas obiektów mogą być punktem wyjścia interesujących badań z dziedziny estetyki i psychologii.

W podsumowaniu autorki wyrażają przekonanie, że w erze komputerów, niezależnie od tego jaki ma się pogląd na celowość stosowania komputerów do badań estetycznych, dobrze jest uświadomić sobie jakie możliwości stwarza technika cyfrowa.

LITERATURA

[1] Birkhoff G.D.: Aesthetic Measure. Canbridge Massachusetts Harvard University Press, 1933
[2] Grabska E.: Computer Aesthetic Evaluation of Polygon Forms. International Symposium on „Structure in Mathematical Theories". San Sebastian, 1990
[3] Jaśkowski S.: Matematyka ornamentu. PWN, Warszawa, 1957
[4] Polarska A.: Automatyczne badanie geometrycznych własności wieloboków – część I. Praca magisterska, Uniwersytet Jagielloński, Katedra Informatyki, Kraków 1990
[5] Sukiennik A.: Automatyczne badanie geometrycznych własności wieloboków – część II. Praca magisterska, Uniwersytet Jagielloński, Katedra Informatyki, Kraków 1990.

Wychodząc naprzeciw zapotrzebowaniu środowiska, redakcja nasza oferuje łamy INFORMATYKI na bezpłatne zamieszczanie anonsów o imprezach informatycznych organizowanych na terenie kraju.

Zwracamy się do organizatorów konferencji, wystaw, konkursów i sympozjów z prośbą o przesyłanie do redakcji krótkich (pół strony maszynopisu) informacji o planowanych imprezach. Oczekujemy na zgłoszenia.

PROMOCYJNA 80% OBNIŻKA CEN!

LOGOTEC® Engineering

Oferujemy

licencjonowane, polskie wersje językowe zachodnioniemieckich systemów wspomagania projektowania

LOGOCAD – mechanika ELOCAD – elektrotechnika

oprogramowanie pomocnicze

pakiety ponad 100 polskich (PN) i niemieckich (DIN) norm mechanicznych (śruby, nakrętki, podkładki, pierścienie, zawleczki, sworznie, sprężyny, łożyska toczne itp.)

inżynierskie programy obliczeniowe:

(wały, belki, ramy, kraty, ruszty, płyty, powłoki, płaski stan naprężeń i płaski stan odkształceń, przekroje cienko- i pełnościenne, obliczenia i optymalizacja kół zębatych)

programy do obsługi baz danych

UNIP2 – uniwersalny generator wydawnictw i programów standardu dBase i Clipper
nagrodzony MikroLaurem '88 i '90
ARCHIW – archiwum rysunków i projektów
MOL(książkowy) – do kompleksowej obsługi biblioteki i wypożyczalni książkowej

doradztwo i szkolenia

z zakresu zastosowania naszych systemów CAD, kursy obsługi naszego oprogramowania, pomoc we wdrożeniu i eksploatacji programów

sprzęt

kompletne stanowiska dla naszych systemów, digitizery, monitory i karty graficzne dla różnych systemów CAD.

Jesteśmy do państwa dyspozycji

LOGOTEC ENGINEERING
Zakład Rozwoju Systemów CAD
Nasz nowy adres: 40-028 KATOWICE, ul. Francuska 70
tel.: 57-46-26, 57-23-76, 51-17-47, teleks 0312606

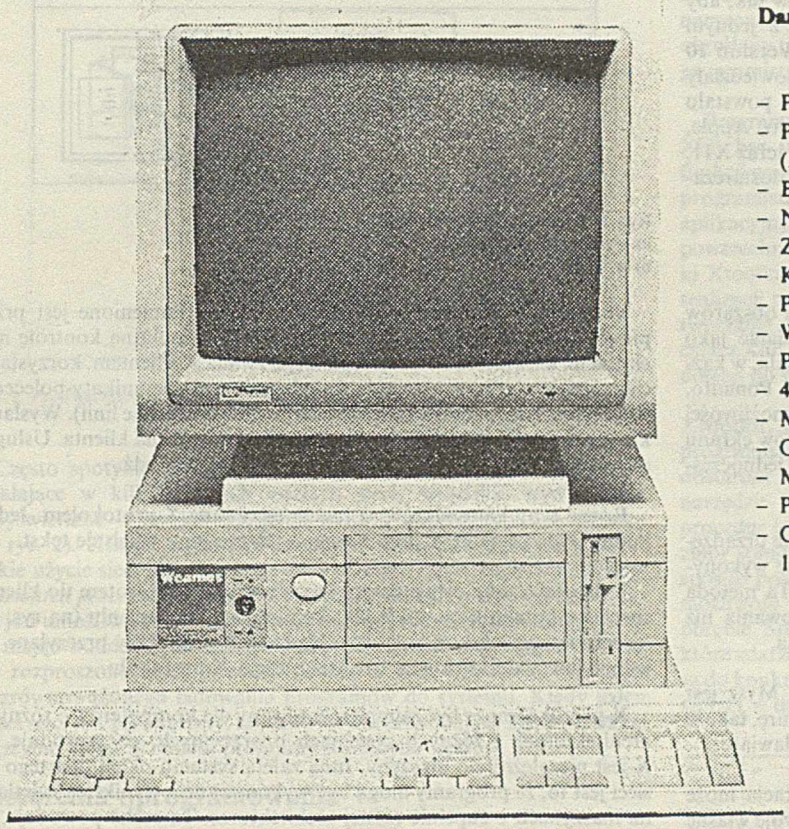
0/21/90

KOMPUTER
Z WBUDOWANĄ PRZYSZŁOŚCIĄ

Komputer zaprojektowany przez czołowego światowego producenta mikrokomputerów – firmę ADVANCED LOGIC RESEARCH i wyprodukowany przez firmę WEARNES TECHNOLOGY.

Komputer, którego możliwości i cena oszałamiają konkurencję.

Komputer, który nie zestarzeje się, dzięki możliwości rozbudowy do architektury 386SX i 486.



Dane techniczne:

SERIA WEARNES BOLDLINE „M"

- Procesor 80286 – 12.5 Mhz
- Pamięć 1 MB RAM, możliwość rozbudowy do 16 MB (5 MB na płycie głównej)
- BIOS Phoenix
- Napęd dyskietek 3,5" 1,44 MB
- Zasilacz 110 Watt
- Klawiatura 101 klawiszy
- Podstawka dla koprocesora matematycznego 80287
- Wbudowany sterownik dyskowy z przeplotem 1:1
- Port szeregowy i równoległy
- 40 MB dysk sztywny
- Możliwość korzystania z EMS 4.0
- Obudowa typu „compact"
- Miejsce na dwa napędy 5,25" o wysokości 1/2
- Podręcznik i dyskietka z programem konfiguracyjnym
- Opcjonalna rozbudowa do 386SX i 486
- 12-miesięczna gwarancja

BOLDLINE COMPUTERS
GRUPA MICOMP-TECH
Biuro Informacji Techniczno-Handlowej
ul. Astrów 7, 40–045 Katowice
telefon i telefaks: 518-628
teleks: 315687 COMP PL

DYSTRYBUTORZY:
PTH „TECHMEX"
43–300 Bielsko-Biała
ul. M. Curie Skłodowskiej 13
tel.: 42-198, 47-555, telefaks: 47-624,
teleks: 35325

O/22/90

Okna w UNIX-ie: System X i jego otoczenie

Od 1988 roku trwa konkurencja pomiędzy OSF[1] a UI[2] w opracowaniu otwartego systemu operacyjnego (p. INFORMATYKA nr 9, 1990). Na początku 1989 roku OSF wprowadziło, jedyny jak do tej pory, produkt – OSF/Motif – rozbudowane graficzne środowisko użytkownika. Od jesieni 1988 roku firmy AT & T i Sun dostarczają własne środowisko graficzne – OPEN LOOK. U podstaw obydwu produktów leży X Windows System (w skrócie X). Wydaje się, że system X będzie w najbliższym czasie rzeczywistym standardem, na którym opierać się będą najbardziej rozpowszechnione środowiska graficzne.

System X Windows powstał w MIT podczas rozpoczętych w 1984 roku prac nad projektem Athena. Celem projektu było zintegrowanie oprogramowania dostarczanego przez różnych producentów tak, aby pracując na różnych maszynach mogło współpracować z jednym systemem okien. Opublikowany w 1986 roku **X Windows Version 10 Release 4** zdobył dużą popularność, a DEC i HP zapowiedziały handlową implementację systemu. W 1988 roku w MIT powstało konsorcjum (zwane X-konsorcjum) z udziałem firm Apollo, Apple, AT & T, DEC, HP, Sun i innych. Ustalono, że X Windows, teraz X11, będzie głównym standardem wejścia–wyjścia w systemach dostarczanych przez te firmy.

Wejście–Wyjście

Podobnie jak inne systemy okien, X dzieli ekran na wiele obszarów wejścia–wyjścia zwanych oknami. Każde z nich może działać jako niezależny wirtualny terminal. Przy użyciu emulatora terminala, w każdym oknie mogą być uruchamiane zwykłe aplikacje tekstowe. Ponadto, w oknie można uruchamiać aplikacje wykorzystujące możliwości graficzne – aplikacja może mieć dostęp do wszystkich punktów ekranu w granicach własnego okna. Aplikacja może także korzystać jednocześnie z wejścia–wyjścia tekstowego i graficznego.

Podstawowym urządzeniem wejściowym jest mysz lub inne urządzenie wskazujące. Mysz umożliwia użytkownikowi sterowanie wykonywania aplikacji często w ogóle bez używania klawiatury. Ta metoda korzystania z programów jest często łatwiejsza do opanowania niż tradycyjne sterowanie z klawiatury – jest bardziej intuicyjna.

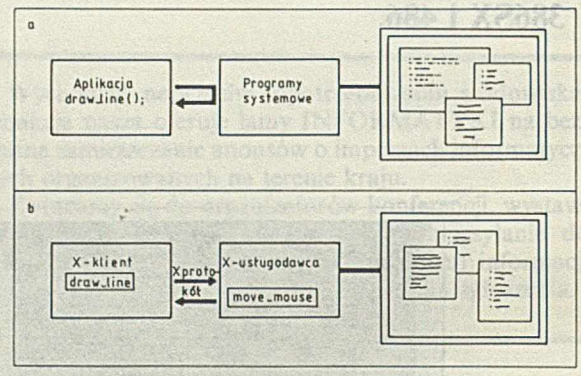
Oczywiście X również udostępnia wejście z klawiatury. Mysz jest używana do wskazania, które okno ma przydzieloną klawiaturę, tak, że tylko jedna aplikacja jednocześnie może dostać wejście z klawiatury.

W X, jak w wielu innych systemach okien, każda aplikacja może korzystać z wielu okien – każda część aplikacji może mieć swoje własne osobne okno. Takie rozwiązanie upraszcza sterowanie wejścia–wyjścia w kodzie aplikacji a jednocześnie pozwala na wizualne wyróżnienie poszczególnych funkcji aplikacji. Aplikacja lub użytkownik może zmienić rozmiary okien, przesuwać je, otwierać oraz zamykać.

Usługodawca, klient, protokół

Wiele z powyższych cech charakteryzuje również inne systemy okien. Tym, co zrobiło się z X standard jest fakt, że X jest oparty na protokole sieciowym – zdefiniowanym zbiorze poleceń i odpowiedzi – w miejsce specyficznych dla tradycyjnych rozwiązań wywołań procedur.

W tradycyjnych systemach, jeżeli aplikacja korzysta z wyjścia graficznego, na ogół wywołuje odpowiedni podprogram biblioteczny lub systemowy. Podprogram ten wykonuje zadanie (na rys. 1a – rysowanie linii) i kiedy zadanie jest wykonane, sterowanie wraca do aplikacji.



Rys. 1. Wyjście graficzne
a) w systemach tradycyjnych
b) w systemie X

W systemie X oprogramowanie systemowe zamienione jest przez proces zwany X-usługodawcą. Proces ten ma absolutną kontrolę nad ekranem, klawiaturą i myszą. Aplikacja, zwana X-klientem, korzystająca z wyjścia graficznego wysyła do usługodawcy komunikaty-polecenia wykonania konkretnego zadania (na rys. 1b – rysowanie linii). Wysłanie komunikatu do usługodawcy nie wstrzymuje działania klienta. Usługodawca może (ale nie musi) wysłać zwrotną odpowiedź.

Różne typy komuniktów są razem nazywane X-protokołem. Jeden komunikat rysuje linię, inny okrąg, a jeszcze inny wypisuje tekst.

Z drugiej strony usługodawca może wysyłać z powrotem do klienta specjalne komunikaty, takie jak: komunikaty o zdarzeniu (na rys. 1b – ruch myszy) lub komunikaty o błędach. Komunikaty przesyłane od usługodawcy do klienta są również częścią X-protokołu.

X-protokół może być zaimplementowany dla komputerów o różnych architekturach i różnych systemach operacyjnych, co powoduje, że X jest niezależny do maszyny. Inną zaletą systemu okien opartego na sieci jest to, że programy mogą być wykonywane a wyniki wyświetlane na maszynach o zupełnie różnej architekturze. Ta cecha powoduje, że system X Windows może tworzyć sieć współpracujących ze sobą różnych komputerów. Na przykład, skomplikowana obliczeniowo aplikacja może być wykonywana na superkomputerze, ale pobiera dane i wypisuje wyniki na stację roboczą poprzez sieć.

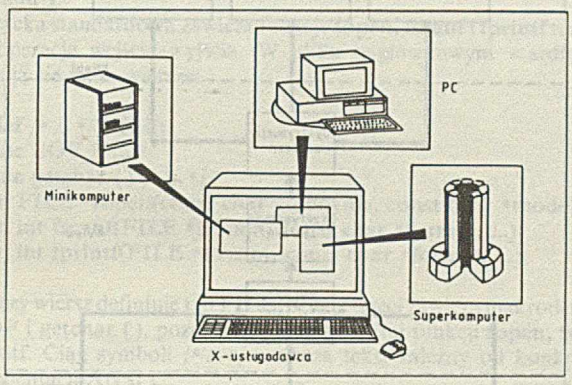
Usługodawca wykonuje następujące zadania:
- Umożliwia wyświetlanie wielu klientom. Usługodawca może odmówić dostępu klientom uruchamianym na pewnych maszynach;
- Interpretuje komunikaty sieciowe od klientów i reaguje na nie. Te komunikaty są określane jako żądania. Pewne komendy nakazują obsługującemu zrobienie dwuwymiarowego rysunku lub przesunięcie okna, podczas gdy inne proszą go o informacje dotyczące – na przykład – rozmiaru okna;
- Przesyła wejście użytkowników do klientów, wysyłając komunikaty sieciowe określane jako zdarzenia, które reprezentują naciśnięcie klawi-

[1] Open Software Foundation – Konsorcjum do opracowywania i promocji przemysłowych standardów oprogramowania, powołane przez firmy IBM, DEC, Bull i Hewlett-Packard.
[2] UNIX International – niedochodowa korporacja firm komputerowych wiążąca nadzieje z przyszłymi wersjami UNIX – System V.

sza klawiatury lub przycisku myszy, przesunięcie myszy itd.;
● Przechowuje złożone struktury danych, takie, jak na przykład kroje znaków. Przechowywanie tych struktur przez usługodawcę redukuje liczbę danych, które musiałyby być przechowywane przez każdego klienta i liczbę danych, które musiałyby być przesyłane przez sieć.

W X termin ekran nie musi oznaczać ekranu jednego monitora. Usługodawca może kontrolować więcej niż jeden fizyczny ekran. Np. jeden usługodawca może zarządzać ekranami monitora kolorowego i monitora monochromatycznego dla użytkownika, który chce śledzić aplikację jednocześnie na obydwu typach ekranu.

Ścieżka komunikacyjna między klientem i usługodawcą nazywa się połączeniem. Kilku klientów może być podłączonych do jednego usługodawcy. Klienci mogą być uruchomieni na tej samej maszynie co usługodawca, o ile maszyna nie jest wielozadaniowa, albo klienci mogą pracować na różnych maszynach w sieci. W każdym przypadku X-protokół jest używany przez klienta do wysyłania żądań, do rysowania grafiki lub do zapytania usługodawcy o informację, a z drugiej strony przez usługodawcę do wysłania użytkownikowi danych lub odpowiedzi na żądania. Cała komunikacja od klienta do usługodawcy i w drugą stronę odbywa się przez X-protokół.



Rys. 2. Przetwarzanie rozproszone

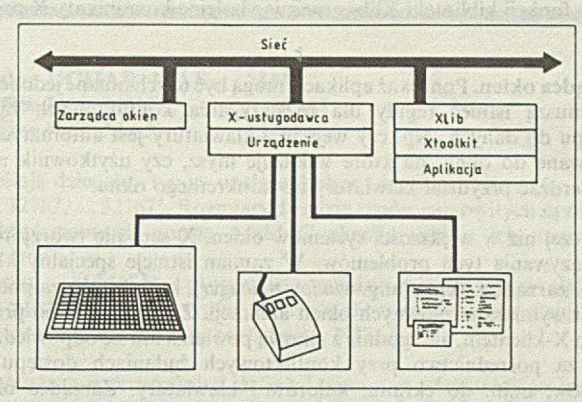
Często spotyka się sytuację, w której użytkownik ma programy działające w kilku różnych węzłach sieci, wszystkie wywoływane i wyświetlane w swoich własnych oknach na pojedynczym ekranie (p. rys. 2). Klienci mogą być uruchamiani na zdalnych maszynach. Takie użycie sieci jest określane jako przetwarzanie rozproszone (ang. *distributed processing*). Najważniejszym powodem takiego przetwarzania jest umożliwienie wyjścia graficznego systemom o dużej mocy, które nie mają wbudowanych umiejętności graficznych. Jednakże przetwarzanie rozproszone może również pomóc w rozwiązywaniu problemu niezrównoważonego ładowania programów do systemu. Kiedy jeden węzeł sieci jest przeciążony, użytkownicy uruchamiający klientów na tej maszynie mogą przenieść programy do innych węzłów.

Hierarchia oprogramowania

Ponieważ X nie został ograniczony do jednego języka, systemu operacyjnego lub środowiska użytkownika, jest wiele sposobów pisania aplikacji. Jedynym wymaganiem stawianym takiej aplikacji jest to, żeby generowała i odbierała komunikaty X-protokołu. Na rys. 3 pokazana jest warstwowość oprogramowania używanego w X-aplikacjach.

Xlib. Aby X-klient mógł komunikować się z X-usługodawcą, musi umieć wygenerować polecenia X-protokołu dla transmisji do usługodawcy oraz rozkodować komunikaty, które otrzymuje od usługodawcy. Tworzenie i rozkodowywanie tych poleceń może być niewygodne i dlatego dostarczana jest biblioteka nazywana Xlib. Xlib jest – z reguły – najniższą warstwa oprogramowania używaną do komunikacji z usługodawcą. Biblioteka ta składa się ze zbioru funkcji napisanych w języku C. Większość z nich generuje pojedyncze polecenia X-protokołu, chociaż są i takie, które generują po kilka poleceń.

Na przykład (por. rys. 4a), jeżeli X-klient używa funkcji *XDrawLine*, to wywołuje odpowiedni kod w Xlib, który tworzy polecenie *PolySegment*. Polecenie *PolySegment* przesyłane jest następnie do X-usługodawcy.

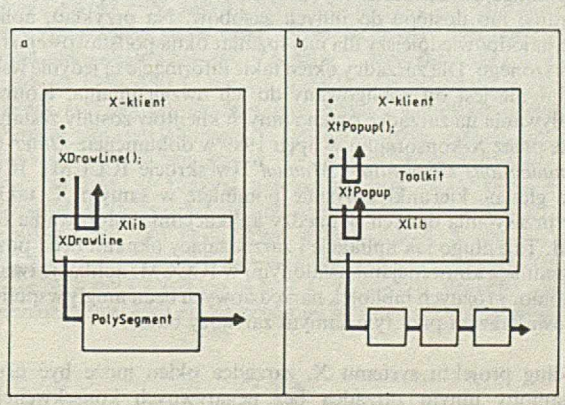


Rys. 3. Warstwowość oprogramowania

Należy zaznaczyć, że biblioteka Xlib nie narzuca X-aplikacjom żadnego specyficznego wyglądu (ang. *look and feel*) na ekranie – składa się ona jedynie z podstawowych poleceń, takich jak: utworzenie okna, narysowanie linii, wypisanie tekstu itd. Wygląd aplikacji jest głównie określany przez inną bibliotekę programów – Xtoolkit – zbiór narzędzi programistycznych.

Program X-klienta może być napisany przy pomocy funkcji biblioteki Xlib bez użycia żadnych innych programów bibliotecznych. Lista funkcji wykonywanych przez Xlib jest jednak zbyt szeroka tak, że gdyby programista musiał włączać bezpośrednio wszystkie elementy, program aplikacyjny byłby zbyt duży. Dlatego zmodularyzowano najbardziej powszechne funkcje w bibliotece Xtoolkit. Jednakże, używanie biblioteki Xtoolkit nie wyklucza wywołania bezpośrednio funkcji z Xlib dla realizacji pewnych zadań, takich jak na przykład rysowanie – na ogół tworzenie grafiki jest bardziej efektywne przy użyciu procedur rysujących z Xlib. Poza tym, jedynie funkcje biblioteki Xlib umożliwiają pełen dostęp do możliwości X-protokołu.

Xtoolkit. Jest to co najmniej pół tuzina bibliotek narzędziowych dla programistów dostępnych dla systemu X Windows. Jest ciekawe, że Sun dostarcza w swoim systemie jednocześnie dwie równoważne biblioteki narzędzi: XOLToolkit oraz XView. Obydwie z nich są zbiorem procedur języka C. Biblioteka narzędzi ma zadanie dostarczyć bogaty zbiór standardowych obiektów używanych w aplikacjach oraz prosty sposób posługiwania się nimi. Każdy zdefiniowany obiekt musi mieć możliwość interpretowania tych zdarzeń, które wystąpiły w jego obrębie. Środowisko tworzone przez bibliotekę narzędziową decyduje, które zdarzenia są interpretowane przez środowisko, a które przesyłane są do konkretnych obiektów. Obiekty oraz obsługa zdarzeń opisana jest szerzej w dalszej części artykułu na przykładzie biblioteki XView.



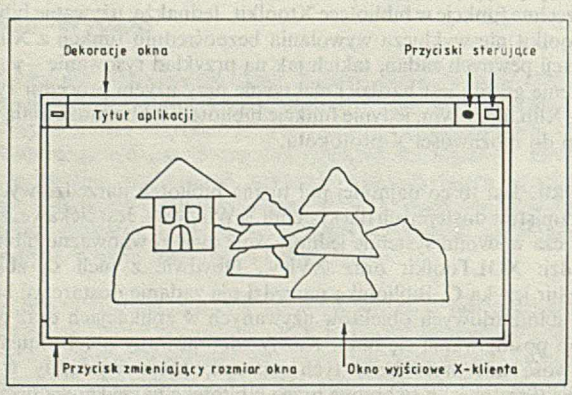
Rys. 4. Generowanie komunikatów przy użyciu biblioteki
a) Xlib
b) Xtoolkit

Biblioteki narzędzi korzystają z funkcji dostarczanych przez bibliotekę Xlib. Na rys. 4b przedstawiony jest przepływ sterowania na przykładzie tworzenia okna – funkcja biblioteki Xtoolkit wywołuje szereg funkcji biblioteki Xlib, generując kolejne komunikaty X-protokołu.

Zarządca okien. Ponieważ aplikacje mogą być uruchamiane jednocześnie, muszą istnieć reguły dla rozstrzygania konfliktowych żądań dostępu do danych. Np. czy wejście z klawiatury jest automatycznie kierowane do okna, na które wskazuje mysz, czy użytkownik musi potwierdzać przydział klawiatury do konkretnego okna.

Inaczej niż w większości systemów okien, X sam nie tworzy reguł rozwiązywania tych problemów. W zamian istnieje specjalny klient zwany **zarządcą okien** (ang. *window manager*), który zarządza położeniem i wymiarami głównych okien aplikacji. Zarządca jest po prostu innym X-klientem, ale zgodnie z umową powierza mu się odpowiedzialność za pośrednictwo przy konfliktowych żądaniach dostępu do zasobów, m.in. do ekranu, kolorów i klawiatury. Zarządca okien pozwala użytkownikowi przesuwać okna po ekranie, zmieniać ich rozmiar i zwykle rozpocząć wykonywanie nowej aplikacji. Zarządca również określa wiele z wizualnego zachowania się systemu okien – czy okna mogą zachodzić na siebie, czy mają być poukładane jedno obok drugiego itp.

Zarządca okien zwykle nakłada kilka dekoracji dookoła każdego okna wyjściowego dla X-klienta m.in. przyciski zmiany rozmiaru i pozycji okna oraz pasek zawierający tytuł okna. Zadaniem zarządcy staje się wtedy zmiana rozmiaru, przesuwanie, zmiana układu okien zgodnie z życzeniem użytkownika przy użyciu myszy kierującej wskaźnik na daną część dekoracji okna lub wyboru danej funkcji z menu zarządcy okien. Na rys. 5 przedstawione są przykładowe dekoracje okna.



Rys. 5. Przykładowe okno X-klienta

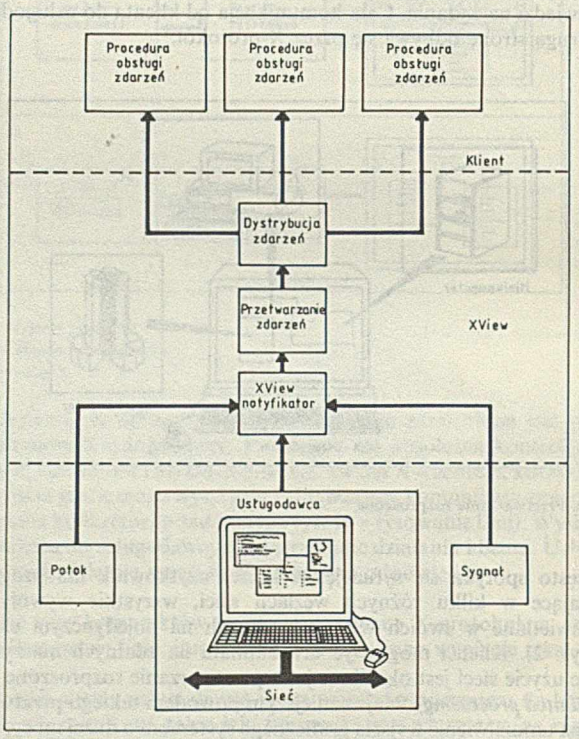
Wymagane jest, aby aplikacje udzielały zarządcy pewnych informacji, które pomagają mu pośredniczyć przy konfliktowych żądaniach miejsca na ekranie lub dostępu do innych zasobów. Na przykład, aplikacja podaje najodpowiedniejszy dla niej rozmiar okna podstawowego i okna powiększonego. Dla zarządcy okien takie informacje są jedynie wskazówkami – nie jest on zobligowany do ich uwzględniania. Konwencje oddziaływania na zarządcę okien i innych klientów zostały zestandaryzowane przez X-konsorcjum w lipcu 1989 w dokumencie *„Inter-Client Communications Conventions Manual"* (w skrócie ICCCM). ICCCM podaje główne kierunki umyślnie pominięte w samym X, takie jak reguły przesyłania danych pomiędzy aplikacjami, kolorowania obszarów itd. Tak długo jak aplikacje i zarządzający oknami będą postępować zgodnie z konwencjami ustalonymi w ICCCM, aplikacje tworzone przy pomocy różnych bibliotek narzędziowych będą mogły współistnieć i pracować razem przy tym samym zarządcy okien.

Według projektu systemu X, zarządca okien może być usunięty i zastąpiony innym zarządcą bez negatywnych konsekwencji dla X-klientów wyświetlających wyjście na ekranie! Stare dekoracje okien są usuwane z ekranu, a na ich miejsce pojawiają się nowe dekoracje wprowadzone przez nowego zarządcę okien.

Zauważmy, że zarządca okien tworzy jedynie *wygląd* (ang. *look and feel*) X-klienta w odniesieniu do jego dekoracji okna. Cokolwiek X-klient wypisuje w swoim oknie, jest to całkowicie niezależne od zarządcy okien.

Obiekty i obsługa zdarzeń w aplikacjach XView

Zdarzenie w systemie X – struktura danych wysyłana przez usługodawcę – opisuje to, co się właśnie zdarzyło, ale tylko to, co jest interesujące dla danej aplikacji. Źródłami zdarzenia są: działanie użytkownika, system okien, system operacyjny i programy aplikacyjne. Np. wciśnięcie przez użytkownika klawisza na klawiaturze lub przycisku na myszy generuje zdarzenie, przesunięcie okna również generuje zdarzenie, ale tylko wtedy, gdy zmieniają się widoczne części okien innych aplikacji. Zadaniem usługodawcy jest rozdzielenie zdarzeń do różnych okien na ekranie. W modelu tworzonym przez View, pomiędzy usługodawcą a aplikacją jest mechanizm przesyłania zdarzeń zwany notyfikatorem (ang. *Notifier*) – por. rys. 6.



Rys. 6. Notyfikator

Typowe aplikacje pisane przy pomocy XView składają się z dwóch części. Pierwsza zawiera definicje wszystkich obiektów, z których aplikacja będzie korzystać. Obiektami mogą być: okna i podokna (tekstowe, graficzne i mieszane), menu, różne przyciski, potencjometry, suwaki służące do przesuwania obrazu nie mieszczącego się w oknie i wiele innych. Następnie, w drugiej części aplikacji, sterowanie przekazywane jest do Xview. Od tej pory notyfikator automatycznie przekazuje zdarzenia do utworzonych obiektów.

Wiele ze zdarzeń obsługiwanych jest przez same obiekty – aplikacja nie musi się nimi w ogóle zajmować. Dla każdego obiektu można zdefiniować specjalną obsługę wybranych zdarzeń albo jedynie dodatkowe czynności wykonywane po zanotowaniu danego zdarzenia. Przez odpowiadanie na zdarzenia aplikacja wykonuje swoje zadania.

Notyfikator nie tylko wysyła zdarzenia generowane przez usługodawcę – dostępne są zdarzenia także z innych źródeł. Aplikacja może obsługiwać wiele przerwań generowanych przez system operacyjny.

dokończenie na s. 26

O standardzie ANSI języka C /3/

Jednym z głównych powodów atrakcyjności języka C była zawsze obszerna biblioteka funkcji standarodowych realizujących w efektywny sposób operacje wejścia–wyjścia, operacje związane z przydziałem pamięci, obsługę przerwań itd.

Biblioteka funkcji standardowych

Norma ANSI języka C specyfikuje minimalny zbiór funkcji wchodzących w skład standardowej biblioteki C. Przyjęto zasadę, że z każdą funkcją biblioteczną związana jest jej deklaracja w odpowiednim pliku tekstowym zwanym plikiem nagłówkowym. Oprócz deklaracji funkcji pliki nagłówkowe mogą zawierać makrodefinicje, deklaracje pewnych zmiennych globalnych i definicje typów struktur danych, na których operują funkcje. Standard ANSI specyfikuje zbiór makrodefinicji i typów struktur danych wchodzących w skład odpowiednich plików nagłówkowych.

Przykład 1
Biblioteka standardowa zawiera funkcje **fopen, fscanf** i **fprintf** realizujące operacje wejścia–wyjścia. W pliku nagłówkowym **<stdio.h>** znajdują się m.in. wiersze:

```
typedef /* ... */ FILE;
#define EOF /* ... */
#define getchar () /* ... */
extern FILE *fopen(const char *filename, const char *mode);
extern int fscanf(FILE *stream, const char *format, ...);
extern int fprintf(FILE *stream, const char *format, ...);
```

Pierwszy wiersz definiuje typ **FILE**, drugi i trzeci zawiera makrodefinicje **EOF** i **getchar ()**, pozostałe są deklaracjami funkcji **fopen, fscanf** i **fprintf**. Ciąg symboli /* ... */ oznacza tekst zależny od konkretnej implementacji ANSI C.

Istotnym i nie do końca rozwiązanym problemem jest kwestia kolizji nazw nadawanych przez programistę funkcjom, zewnętrznym zmiennym globalnym (atrybut **extern**), typom i makrodefinicjom z nazwami funkcji bibliotecznych oraz nazwami typów i makrodefinicji występującymi w plikach nagłówkowych. Norma zaleca jedynie unikanie nazw zaczynających się od znaku podkreślenia.

Opis standardowej biblioteki C specyfikuje minimalny zbiór makrodefinicji, definicji typów oraz deklaracji zmiennych globalnych i funkcji związanych z każdym z plików nagłówkowych, opisując jednocześnie semantykę każdej z funkcji. Standard ANSI wymienia 15 plików nagłówkowych:

<assert.h>	**<locale.h>**	**<stddef.h>**
<ctype.h>	**<math.h>**	**<stdio.h>**
<errno.h>	**<setjmp.h>**	**<stdlib.h>**
<float.h>	**<signal.h>**	**<string.h>**
<limits.h>	**<stdarg.h>**	**<time.h>**

Niektóre z wymienionych wyżej plików nie są związane z żadnymi szczególnymi pakietami funkcji. Plik **<limits.h>** zawiera makrodefinicje określające minimalne i maksymalne wartości związane z typami całkowitymi. Plik **<float.h>** specyfikuje wartości związane z reprezentacją i arytmetyką liczb zmiennoprzecinkowych. W pliku **<stddef.h>** można znaleźć definicje najczęściej używanych typów i „klasyczne" makrodefinicje.

Przykład 2
W pliku nagłówkowym **<limits.h>** makrodefinicje:

```
#define UCHAR_MAX   255U
#define INT_MIN     – 32767
#define INT_MAX     32767
```

określają dziedziny typów **unsigned char** i **int** jako zbiory {0, ..., 255} i {– 32767, ..., 32767}. Rozmiary dziedzin typów całkowitych są zależne od implementacji; norma ANSI C określa jedynie ich minimalne rozmiary.
W pliku **<stddef.h>** programista znajdzie między innymi definicję typu **ptrdiff_t** (jeden z typów całkowitych ze znakiem określający typ różnicy wskaźników) i makrodefinicję **NULL**.

Pełna lista makrodefinicji, definicji typów oraz deklaracji zmiennych globalnych i funkcji zawarta w standardzie ANSI C liczy około 200 pozycji. Próba najbardziej nawet lakonicznego opisu przekroczyłaby znacznie ramy niniejszego artykułu. Niżej wymienimy tylko dziedziny zastosowań pakietów funkcji związanych z poszczególnymi plikami nagłówkowymi:

<assert.h> – diagnostyka programów C,
<ctype.h> – klasyfikacja i taransformacja znaków,
<errno.h> – diagnostyka standardowych błędów,
<math.h> – zmiennoprzecinkowe funkcje matematyczne,
<setjmp.h> – niestandardowe wywoływanie funkcji,
<signal.h> – obsługa sygnałów,
<stdarg.h> – obsługa funkcji o zmiennej liczbie parametrów,
<stdio.h> – standardowe wejście–wyjście,
<stdlib.h> – całkowite funkcje matematyczne, przydział pamięci, sortowanie, wyszukiwanie, konwersja tekstów na stałe liczbowe
<string.h> – obsługa ciągów znaków,
<time.h> – funkcje obsługi daty i czasu.

Zakładając, że Czytelnicy zetknęli się już w dostępnych implementacjach z wymienionymi plikami nagłówkowymi i znają mniej lub bardziej szczegółowo przeznaczenie większości związanych z nimi pakietów funkcji, nie będziemy się nimi dłużej zajmować. Na koniec przedstawimy tylko krótko przeznaczenie pakietu **<locale.h>**.

Pakiet ten został wprowadzony do standardu stosunkowo niedawno i ma silny związek z tendencją do internacjonalizacji języka C. Najkrócej mówiąc pakiet **<locale.h>** ma dostosować działanie niektórych funkcji standardowych do specyficznych dla danego kraju wymagań dotyczących alfabetu, sposobu zapisu daty czy sposobu zapisu liczb zmiennoprzecinkowych.

Czytelnika mającego doświadczenie w pisaniu programów w języku C zapewne zdziwi brak pakietu funkcji obsługujących niebuforowane wejście–wyjście: **open, close, read, write** i **lseek** oraz związanego z tymi funkcjami pliku nagłówkowego **<fcntl.h>**. Podobnie nigdzie w opisie standardowej biblioteki C nie ma wzmianki o takich funkcjach, jak **ioctl, execl, stat** czy **popen**. Standard ANSI C wszystkie wymienione wyżej funkcje traktuje jako funkcje systemowe (ang. *system calls*), a nie jako funkcje biblioteczne języka C. Oczywiście dyskusyjnym może wydawać się fakt, że funkcja **getenv** znalazła się w klasie funkcji bibliotecznych związanych z plikiem nagłówkowym **<stdlib.h>**, a funkcja **putenv** nie. Na pocieszenie należy dodać, że norma ANSI C specyfikuje minimalną standardową bibliotekę C i najprawdopodobniej w każdej implementacji ANSI C Czytelnik znajdzie zarówno funkcje systemowe, jak i nie wymienione w standardzie pliki nagłówkowe.

* * *

Niektórzy z Czytelników mogą się czuć zawiedzeni takim, a nie innym wyborem informacji o standardzie ANSI C zamieszczonym w tym

UNIX jest zastrzeżonym znakiem towarowym AT & T.

artykule. Nie było zamiarem autora przekazanie Czytelnikom formalnego opisu standardu ANSI języka C oraz pełnej specyfikacji funkcji wchodzących w skład biblioteki standardowej; szczupłe ramy **Informatyki** na to nie pozwalały. Suchy, formalny opis języka i specyfikacja funkcji bibliotecznych zajmuje w dostępnej autorowi wersji raportu przeszło 100 stron.

Może warto pokusić się o szybkie wydanie przetłumadzonego na język polski standardu ANSI C?

LITERATURA

[1] Butzen F.: Porting to ANSI C, Unix World, June 1989
[2] Draft Proposed American National Standard for Information Systems – Programming Language C, February 1989
[3] Hersee S.A., Knopoff D.: An ANSI Standard for the C Language, BYTE, March 1986
[4] Kernigham B.W., Ritchie D.M.: Język C, WNT, 1987.

WŁADYSŁAW MAJERSKI

/mail

W ostatnim zeszłorocznym numerze prosiliśmy o nadsyłanie opinii o naszym dziale. Oto fragment jednego z listów:

Szanowni Państwo,

(...) Chciałbym wyrazić uznanie za podjęcie (w końcu) tego tematu. Z wiadomych powodów UNIX nie był i nie jest zbyt popularnym systemem w Polsce, ale wydaje się, że czas najwyższy ten stan rzeczy zmienić. Ktoś mógłby zapytać po co rozwijać dział, który być może jest interesujący dla niewielkiej grupy czytelników. Natychmiast odpowiedziałbym, że ten argument jest zasadny w przypadku wielu innych rodzimych pism „informatycznych", ale nie w przypadku „Informatyki"! Wszak jest to jedyne pismo profesjonalne – od informatyków dla informatyków. Jeśli nawet (choć nie jestem pewien) spora część czytelników to nie zawodowi informatycy, pozostaje jedynie cieszyć się, że sięgają po profesjonalne pismo.

(...) Na koniec kilka moich prywatnych sugestii a propos zawartości tego działu.
Po pierwsze – wydaje mi się, że byłoby celowe choć krótkie porównanie istniejących mutacji/implementacji UNIX-a (AT & T, Berkeley, SCO, Interactive, etc.) oraz interpreterów poleceń (C, Bourne, Korn).
Po drugie – chciałbym widywać krótkie i użyteczne, ale nietrywialne skrypty. Zauważyłem, że większość podręczników podaje tylko szkolne przykłady.
Po trzecie – może rozpocząć dłuższy cykl artykułów (5–6) na temat filozofii UNIX-a? Myślę raczej nie o kursie podstawowych komend UNIX-a, ale o elementach budowy systemu – coś w rodzaju skrótu monografii Bacha. Dostęp do odpowiedniej literatury jest raczej niewielki...

Z informatycznym pozdrowieniem
Janusz Kalinowski, Wrocław

Dziękujemy za poparcie i propozycje. W najbliższych numerach planujemy rozpocząć kilka dłuższych cykli, w tym, między innymi, o bezpieczeństwie UNIX-a („Strzeżonego UNIX strzeże"), o systemach rozproszonych i sieciowych („Jak UNIX z UNIX-em") i o UNIX-owych systemach plików („Plik plikowi nie równy"). Porównanie trzech UNIX-owych interpretatorów poleceń to dobry pomysł – chyba nawet mamy już autora. Omówienie budowy UNIX-a w kilku z konieczności krótkich odcinkach to zadanie raczej karkolomne, ale pomyślimy i o tym. Najgorzej z naprawdę niebanalnymi skryptami – w zasadzie do tego powstała rubryka /pomysły, która jednak znacznie lepiej funkcjonowałaby jako rubryka pytań i odpowiedzi. Póki co, zachęcamy do studiowania rozlicznych skryptów systemowych, rozsianych po drzewie katalogów każdego UNIX-a. Czekamy na dalsze uwagi.

autorzy /unixa

Okna w UNIX-ie
System X i jego otoczenie

dokończenie ze s. 24

Może odpowiadać na sygnały, przerwania zegarowe, może być zawiadamiana o nadejściu danych poprzez plik. Można również przesyłać zdarzenia pomiędzy klientami w obrębie jednego procesu. Aplikacja może także wpływać na zmianę działania notyfikatora.

Możliwość rozszerzania systemu X

Przy projektowaniu systemu X przewidziano potrzebę przyszłych rozszerzeń. Kod zawiera zdefiniowany mechanizm do wcielania rozszerzeń, tak że przy dodawaniu kolejnych cech producenci oprogramowania nie są zmuszeni do modyfikacji istniejącego systemu. Rozszerzenie wymaga dodatkowego fragmentu programu po stronie usługodawcy i dodatkowej biblioteki na tym samym poziomie co Xlib po stronie klienta. Po zapytaniu początkowym, którego celem jest sprawdzenie czy część rozszerzenia przy usługodawcy jest zainstalowana, rozszerzenia są używane po prostu jak procedury Xlib i działają na tym samym poziomie.

Wśród rozszerzeń, które są aktualnie rozwijane jest dodatek dla dwu wymiarowych krzywych gładkich, dla grafiki trójmiarowej i dla wyświetlania przy pomocy PostScriptu. Rozszerzenia te mogą być używane w aplikacjach narzędziowych tak samo jak funkcje biblioteki Xlib.

TOMASZ WÓJTOWICZ

LITERATURA

[1] Bowen B.: Open Look vs. Motif: The Battle for GUI Dominance. SunTech Journal, pp. 26–34, December 1990.
[2] Heller D. i in.: The X Window System. O'Reilly & Associates, Inc. 1990
[3] Moore D.: The Migration of the X Window System. BYTE IBM Special Edition, pp. 183–185, Fall 1990.

SPROSTOWANIE

Tegoroczny marcowy numer INFORMATYKI w całości był poświęcony symulatorom lotu. W numerze tym nie podaliśmy informacji, że pierwsze wersje wszystkich artykułów były wygłaszane podczas I Krajowej Konferencji Naukowej pn. Grafika Komputerowa i Przetwarzanie Obrazów '90 (GKPO) w dniach 21–24 maja 1990 r. w Kazimierzu Dolnym.

Przepraszamy Autorów za opublikowanie tekstów bez autoryzacji i wprowadzenie terminologii używanej dotychczas w naszym czasopiśmie.

Jednocześnie wyjaśniamy, że streszczenia artykułów w językach obcych zostały przetłumaczone w redakcji i nie zweryfikowane przez Autorów.

Za powyższe uchybienia gorąco przepraszamy.

Redakcja

POZNAJ SIŁĘ NASZYCH SYSTEMÓW INFORMATYCZNYCH!

JUNISOFTEX Sp. z o.o.
44-100 Gliwice ul. Konstytucji 11
Tel. 31-75-10, 31-90-81 do 88 wewn. 250, 272, telefaks: 31-75-10, teleks 036233

Już od 3 lat oferuje kompleksową komputeryzację przedsiębiorstw, a zwłaszcza:

- profesjonalne nowoczesne systemy informatyczne oparte na kompilatorze języka Turbo Pascal 5.5 firmy BORLAND (typu multiusers), pracujące w sieciach NOVELL 2.15 i 3.1 i wykorzystujące interfejs BTRIEVE amerykańskiej firmy SoftCraft – najefektywniejszy i najszybszy system obsługi dużych i bardzo dużych baz danych (do 4 miliardów rekordów w pojedynczej bazie danych).

Polecamy szczególnie systemy zintegrowane

- wielodostępny system finansowo-księgowy wg trójstopniowej analityki w podstawowym układzie: xxx-xxxx-xxxx-xxxxxx z możliwością graficznej prezentacji wyników,
- wielodostępny system gospodarki materiałowej wykorzystujący w strukturze indeksowej SWW i KTM z automatycznym wyliczaniem cyfry kontrolnej (długość indeksu materiałowego 13 + 3 cyfry),
- wielodostępny system gospodarki wyrobami gotowymi wykorzystujący w strukturze indeksowej SWW i KTM,
- wielodostępny system gospodarki przedmiotami nietrwałymi wykorzystujący w strukturze indeksowej SWW i KTM,
- wielodostępny system środków trwałych wykorzystujący klasyfikację środków trwałych GUS,
- wielodostępny system fakturowania sprzedaży,
- wielodostępny system kadrowo-płacowy.

Wszystkie oferowane systemy są zintegrowane ze sobą zgodnie z zasadami rachunkowości księgowej przedstawionymi w miesięczniku RACHUNKOWOŚĆ nr 9, 1990.

Oferujemy również:

- sprzęt komputerowy (sprawdzony w eksploatacji) firm:
  - ALR (Advanced Logic Research) i Storage Dimensions z USA,
  - AZTECH, WEARNES TECHNOLOGY, NAC i inne z Singapuru,
- dostawę i instalację licencjonowanej sieci Novell 2.15 i 3.1,
- usługi w zakresie opracowywania zakładowych indeksów materiałowych wg klasyfikacji GUS (SWW i KTM).

Organizujemy:

- szkolenia i prezentację systemów u naszych kontrahentów.

Gwarantujemy:

- najwyższą jakość naszych systemów i usług,
- najszybsze w Polsce wdrożenia systemów (od 1 do 3 miesięcy).

W przypadku niedotrzymania przez nas terminów zwracamy 25% poniesionych nakładów za każdy miesiąc zwłoki. Dla klientów zamawiających kompleksowe usługi – znaczne zniżki cenowe.

Poszukujemy niezawodnych

- pośredników i dealer'ów naszych systemów informatycznych (oferujemy bardzo atrakcyjne warunki współpracy),
- dostawców i pośredników dobrego i taniego sprzętu komputerowego.

Z siły naszych systemów skorzystały dotąd następujące firmy:

Huta ŁAZISKA w Łaziskach Górnych	tel. 24-14-06
Walcownie Metali w Czechowicach-Dziedzicach	tel. 52-35-1
Centrala Zaopatrzenia Górnictwa w Bytomiu	tel. 81-92-91
Mikołowska Fabryka Transformatorów w Mikołowie	tel. 26-25-41
BPiKD PREFAMET-ZREMB w Gliwicach	tel. 38-20-51
BPiKD METALCHEM w Gliwicach	tel. 31-56-33
Instytut Informatyki Teoretycznej i Stosowanej PAN w Gliwicach	tel. 31-71-50
Instytut Spawalnictwa w Gliwicach	tel 31-00-11
PP ENERGOROZRUCH w Gliwicach	tel. 37-66-25
Fabryka Palenisk Mechanicznych w Mikołowie	tel. 26-20-02
Zakłady Mechaniczne Górnictwa i Energetyki WIROMET w Mikołowie	tel. 26-23-46
Przędzalnia Bawełny PRZYJAŹŃ w Zawierciu	tel. 22-39
PRT TERMOIZOLACJA w Zabrzu	tel. 71-40-21
Przedsiębiorstwo Usług Informatycznych ETOM w Mikołowie	tel. 57-10-50
Zakład Informatyki i Techniki Komputerowej MEGA w Gliwicach	tel. 31-52-42
Mikołowska Fabryka Maszyn Górniczych MIFAMA w Mikołowie	tel. 26-22-66
Zakłady Produkcji Urządzeń Mechanicznych ELWO w Pszczynie	tel. 30-61
Zakłady Sprzętu Sieciowego i Elektroinstalacyjnego POLAM w Katowicach	tel. 52-80-31
Urząd Miasta w Mikołowie	tel. 26-23-90
Zakłady Elelktromaszynowe CELMA w Cieszynie	tel. 21-58-1
PSS SPOŁEM w Zawierciu	tel. 21-46-17

Chcesz mieć pełną gwarancję i niezawodne efekty komputeryzacji swojego przedsiębiorstwa – współpracuj z firmą JUNISOFTEX!

0/1/91

Nałęcki K.: Dlaczego UNIX nie jest systemem czasu rzeczywistego? INFORMATYKA 1991, nr 6, s. 2 Charakterystyka wymagań, jakie powinien spełniać system operacyjny czasu rzeczywistego oraz uzasadnienie, że wymagań takich nie może spełnić system typu UNIX.	**Nałęcki K.: Why UNIX is not a real-time system?** INFORMATYKA 1991, No. 6, p. 2 Characteristics of conditions which will acomplish a real-time operating system and explanation, that such conditions can not fulfil any UNIX-type system.	**Nałęcki K.: Warum UNIX ist kein Real-Time-System?** INFORMATYKA 1991, Nr. 6, S. 2 Eine Charakteristik von Forderungen, die ein Echtzeitsystem gewährleisten soll, sowie eine Begründung, dass ein UNIX-ähnliches System solchen Forderungen nicht befiedigen kann.
Trojniar W.: Metody implementacji systemów czasu rzeczywistego INFORMATYKA 1991, nr 6, s. 8 Charakterystyka różnych metod implementacji na przykładzie wybranych systemów operacyjnych czasu rzeczywistego.	**Trojniar W.: Implementation methods for real-time systems** INFORMATYKA 1991, No. 6, p. 8 Characteristics of different implementation methods on examples of selected real-time operating systems.	**Trojniar W.: Implementierungsmethoden für Echtzeitsysteme** INFORMATYKA 1991, Nr. 6, S. 8 Eine Charakteristik von verschiedenen Implementierungsmethoden auf Beispielen von ausgewählten Echtzeitbetriebssystemen.
Gizbert-Studnicki P., Karczmarczuk J.: Środowisko do projektowania prostych systemów czasu rzeczywistego – ćwiczenie w programowaniu obiektowym INFORMATYKA 1991, nr 6, s. 12 Charakterystyka języka Smalltalk-80 oraz zalet programowania obiektowego w zastosowaniu do projektowania systemów czasu rzeczywistego, a także omówienie rozwiązania środowiska projektowego dla takich systemów.	**Gizbert-Studnicki P., Karczmarczuk J.: Environment for designing simple real-time systems – an exercise in object programming** INFORMATYKA 1991, No. 6, p. 12 Characteristics of the Smalltalk- 80 language and of object programming advantages in application for real-time systems designing, as well as discussion of design environment solution for such systems.	**Gizbert-Studnicki P., Karczmarczuk J.: Umwelt für Projektierung einfacher Echtzeitsysteme – eine Übung in Objektprogrammieren** INFORMATYKA 1991, Nr. 6, S. 12 Eine Charakteristik der Smalltalk-80-Sprache und der Vorteile von Objektprogrammieren in Anwendungen für Echtzeitprojektierung, sowie eine Besprechung einer Lösung von Projektierungsumwelt für solche Systeme.
Grabska E., Polarska A.: Rola komputera w ocenie estetycznej ksztaltów INFORMATYKA 1991, nr 6, s. 17 Charakterystyka teorii Birkhoffa oraz programu służącego do oceny estetycznej obiektów twórczości artystycznej, a także korzyści jakie można osiągnąć w tym nowym obszarze zastosowań komputera.	**Grabska E., Polarska A.: The part of computer in aesthetic evaluation of shapes** INFORMATYKA 1991, No. 6, p. 17 Charasteristics of the Birkhoff theory and of the program for aesthetic evaluation of artistic objects, as well as underlining of achievable advantages in this new area of computer application.	**Grabska E., Polarska A.: Eine Rolle des Computers in ästhetischer Beurteilung von Gestalten** INFORMATYKA 1991, Nr. 6, S. 17 Eine Charakteristik der Birkhoff-Theorie und des Programmes für ästhetische Beurteilung von Objekten des Künsterschöpfens, sowie eine Verweisung auf Erreichbare Vorteile in diesem neuen Anwendungsbereich des Computers.

Metody implementacji systemów ...

dokończenie ze s. 10

Następny, trzeci element monitora, to procedury obsługi przerwań wbudowane w jądro monitora. Każdy proces może zwrócić się do monitora w celu oczekiwania na wystąpienie określonego przerwania lub przerwań. W chwili wystąpienia przerwania, proces oczekujący zostaje ustawiony jako *„gotowy"* oraz następuje ewentualne przełączenie kontekstu. W czasie pracy monitora wszystkie przerwania są odmaskowane.

Ostatnim elementem monitora są procedury obsługi zegara systemowego. Każdy proces może zwrócić się do monitora w celu oczekiwania na upływ określonego czasu (wielokrotność 1 ms). W chwili zakończenia czasu oczekiwania, proces oczekujący zostaje ustawiony jako *„gotowy"* i następuje ewentualne przełączenie kontekstu.

W czasie pracy monitor użytkuje dla swoich celów następujące zasoby procesora 8051:

- całą pamięć wewnętrzną RAM procesora oraz ok. 800 bajtów pamięci kodu;
- zegar T0, jako zegar systemowy;
- sterownik przerwań (maski przerwań są inicjowane, ale nie zmieniane w czasie pracy monitora).

Monitor nie ma wbudowanych mechanizmów kolejkowych. Przełączenie kontekstu następuje na skutek sygnalizacji semafora, nadejścia przerwania lub upływu określonej liczby kwantów czasu. Przełączenie kontekstu polega na zmianie banku rejestrów i odtworzeniu wskaźnika stosu oraz rejestrów akumulatora, wskaźnika i statusu. Aktywizowany jest **zawsze** gotowy proces o najwyższym priorytecie.

* * *

W artykule przedstawiono niektóre z technik implementacji jądra wielozadaniowego na przykładzie wybranych systemów. Poznanie tych technik przybliża odpowiedź na pytanie, czy dany system nadaje się do użycia w określonej aplikacji. Przed przystąpieniem do stosowania takich systemów, dla aplikacji krytycznych wg kryteriów czasu należy dobrze zrozumieć sens podanych przez projektantów parametrów systemu.

Nowe Książki

WYDAWNICTWA NAUKOWO-TECHNICZNE

George W. Gorsline: Mikrokomputery 16-bitowe. Rodzina INTEL I8086. WNT, Warszawa, 1990, wyd. 1, s. 536. ISBN 83-204-1099-1

Książka jest poświęcona komputerom 16-bitowym. Została napisana przez wykładowcę informatyki na Politechnice w Virginii, przy czym Autor wiele uwagi poświęcił potrzebom praktyków w zakresie projektowania i opracowywania programów dla mikrokomputerów, zwracając uwagę zarówno na aspekty systemowe, jak i sprzętowe.

Książka jest przeznaczona dla użytkowników mikrokomputerów 16-bitowych, studentów kierunków informatycznych na wydziałach politechnicznych i uniwersyteckich oraz uczniów szkół średnich i hobbistów.

Herman Rotermund: WordPerfect. WNT, Warszawa, 1990, wyd. 1, s. 96. ISBN 83-204-1302-8

Książka stanowi opis edytora WordPerfect w wersji 4.2. W pierwszych rozdziałach zawarto podstawowe informacje dotyczące pisania tekstu i poprawiania go, w dalszych – wskazówki dotyczące pracy o charakterze profesjonalnym; dołączono także konkretne makropolecenia.

Książka jest przeznaczona dla każdego użytkownika, zarówno niedoświadczonego, jak i tego, dla którego WordPerfect jest kolejnym stosowanym edytorem tekstu.

Jan Bielecki: Turbo C z grafiką dla IBM PC. WNT, Warszawa, 1990, wyd. 1, s. 404. ISBN 83-204-1101-7

W książce przedstawiono najnowszą implementację języka Turbo C dokonaną przez firmę Bordland International. Dokonano szczegółowego przeglądu konstrukcji języka uzupełniając go licznymi przykładami programów. Dużo uwagi poświęcono rozszerzeniom graficznym języka C, w tym zasadom posługiwania się trybami graficznymi, oraz opisom podprogramów graficznych.

Książka jest przeznaczona dla zainteresowanych programowaniem w języku C, szczególnie dla tych, którzy zamierzają posługiwać się implementacją Turbo C.

PRZEDSIĘBIORSTWO
ZASTOSOWAŃ INFORMATYKI

meditronik

oferuje

UM7106	18 000
UM3481,2,3,4,10,11	8200
UM66T	3600
UM82450	14 000
UM8250B	14 000
UM8253-5	16 500
UM8259A-2	27 300
UM82C11	13 000
UM82C55A	16 500
UM82C8167	19 500
UM6116U-35(ns)	25 500
UM6116-2L	11 500
UM6264-10	20 000
UM62256A-10L	52 000

Nasz adres:
PZI „MEDITRONIK"
00-194 Warszawa, ul. Dzika 4
telefon 635-22-63, 635-22-64
telefaks 635-21-95
teleks 816075 medi pl

0/24/90

Posiadasz maszyny

typu IBM, serii SM, PDP-11, MERA400...,

pracujące

pod nadzorem systemów operacyjnych

UNIX, XENIX, RSX, PC-MOS, MULTILINK, CONCURRENT PCDOS, QNX, CROOK...

To z całą pewnością są ci potrzebne

TERMINALE ALFANUMERYCZNE
MT220

Skontaktuj się z nami.

MICRONET

Zakłady Elektroniczne
"MICRONET"
81-836 Sopot ul. Krasickiego 9
tel. 51-13-17 tlx 051-2876 mcrnt

0/17/90

ZAPRASZAMY

Niklaus Wirth: Modula 2. Warszawa WNT, 1991, wyd. 2 poprawione, s. 228, ISBN 83-204-1243-9

Książka jest monografią języka programowania Modula 2, który powstał w Zurychu pod koniec lat siedemdziesiątych i zdobywa coraz większą popularność na całym świecie. Modula 2 wyrosła z praktycznego zapotrzebowania na ogólny, łatwo realizowany język programowania dla minikomputerów. Modula 2 jest bezpośrednim potomkiem Pascala i Moduli. Głównymi rozszerzeniami tego języka w stosunku do Pascala są: pojęcie modułu, bardziej usystematyzowana składnia, pojęcie procesu (jako klucz do wieloprogramowości), mechanizmy niskiego poziomu oraz typ procedurowy. Modula 2 ułatwia programowanie strukturalne oraz pracę projektową i wdrożeniową w zespołach. Omawiane konstrukcje językowe są w książce ilustrowane licznymi przykładami. Praktyczne wskazówki jak i kiedy stosować określone mechanizmy językowe, pomagają w opanowaniu właściwego stylu programowania. Do książki dołączono Raport zawierający definicję języka.

Książka jest przeznaczona dla programistów, projektantów systemów przetwarzania informacji, pracowników nauki zajmujących się informatyką oraz dla studentów kierunków informatycznych.

Peter P. Silvester: System operacyjny UNIX™. WNT, Warszawa, 1990, wyd. 1, s. 320. ISBN 83-204-1086-x

Książka jest poświęcona systemowi Unix. Jest to popularny i uniwersalny system operacyjny dla wielu komputerów różnych wielkości i typów, m.in. IBM PC. System umożliwia równoczesną pracę interakcyjną wielu użytkowników, zwłaszcza tworzących oprogramowanie lub przetwarzających teksty. Książka zawiera opis maszyny wirtualnej tworzonej przez system dla programów, poleceń sterujących systemem, programów usługowych i kompilatorów.

Książka jest przeznaczona dla szerokiego kręgu osób zainteresowanych tworzeniem oprogramowania.

W serii Podręczna Pamięć Programisty ukazały się:

Agnieszka Mykowiecka – Lotus 1-2-3, wersja 2.0 (wyd. 2)
Krzysztof Kamiński – Turbo Pascal 5.0

TERA Spółka Z O. O.

Przedsiębiorstwo Popierania Postępu TERA Spółka z o.o. uprzejmie informuje, że posiadając kilkuletnie doświadczenie w instalacji systemów wspomagających zarządzanie

o f e r u j e :

- opracowanie projektów systemów,
- optymalny kosztowo i rozwojowo dobór sprzętu i oprogramowania,
- instalację systemu u klienta,
- bezpłatnie przez rok konserwację oprogramowania oraz serwis sprzętu wraz z doradztwem techniczno-eksploatacyjnym,
- szybką dostawę uzupełnień konfiguracji lub sprzętu mikrokomputerowego niezależnego od systemu, w tym mikrokomputerów ALR firmy Wearnes Technology.

Wszelkie dodatkowe informacje uzyskają Państwo codziennie oprócz niedziel w Biurze Handlowym
40–025 Katowice, ul. 27 Stycznia 31/17
tel. (faks): 51-61-65, teleks: 315448 tera pl

PAMIĘTAJ! Instalacje XENIX/NOVELL/PC MOS 386 oraz serwis to nasza specjalność – ZAPRASZAMY, ponieważ czterech lat doświadczeń nigdzie nie kupisz.

0/19/90

MEDCOM

CHCESZ ZABEZPIECZYĆ BAZY DANYCH PRZED UTRATĄ INFORMACJI?

CHCESZ ZAPEWNIĆ PRAWIDŁOWĄ PRACĘ

KOMPUTERA * CENTRALI TELEFONICZNEJ * TELEFAKSÓW

W WARUNKACH ZANIKU NAPIĘCIA ZASILAJĄCEGO

kup ZASILACZE BEZPRZERWOWE

naszej produkcji
UPS – 120 (abonenckie centrale telefoniczne i telefaksy)
UPS – 450
UPS – 850
UPS – 1450

produkcji
AMERICAN POWER CONVERSION:

UPS – 110
UPS – 310
UPS – 400
UPS – 600
UPS – 800
UPS – 1200

produkcji BEST POWER TECHNOLOGY ON – LINE:
500 VA – 18 kVA

Sprzedajemy również

* AKUMULATORY BEZOBSŁUGOWE, HERMETYCZNE FIRMY PANASONIC do zasilania urządzeń alarmowych, jedyne dopuszczone do używania w pomieszczeniach zamkniętych,
* GENERATORY PRĄDOTWÓRCZE FIRMY ENDRESS O MOCY OD 1 kVA DO 350 kVA

Twoje problemy w dziedzinie zasilania awaryjnego rozwiążą specjaliści z firmy MEDCOM.

BIURO
Al. Ujazdowskie 26/39
00-478 Warszawa
tel. 628-93-57, teleks 817060

ZAKŁAD PRODUKCYJNO-USŁUGOWY
ul. Bokserska 53
02-690 Warszawa

0/20/90